INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Interconnection of LANs through ISDN

by

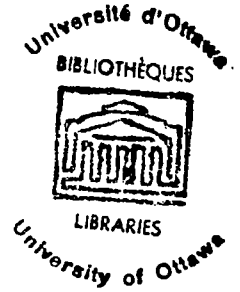
Zhengyu Liu, B.A.Sc.

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfilment of the requirements
for the degree

Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

March 1991



I hereby declare that I am the sole author of this thesis.

I authorize The University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Zhengyu Liu

I further authorize The University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Zhengyu Liu

Contents

Abstract

Network interconnection (internetworking) is an important issue in the area of computer communication. One of the purposes of internetworking is to allow communication to go beyond a single local area network (LAN).

In this thesis, networks interconnection is studied. An internetworking device - (gateway) is proposed to provide the capability of internetwork data transmission between LANs through an ISDN (Integrated Service Digital Network). This gateway is an application-layer program which interfaces with the NetBIOS (Network Input Output System), a widely implemented interface between application programs and LANs in the IBM personal computer world. The NetBIOS interface is provided in both LAN and ISDN environments in a connection-oriented mode. A facility for file transfer and information exchange among LANs through ISDN is developed in such a way that it neither affects the existing network hardware and software available in both LANs and ISDN nor changes the mechanism of communication within a LAN.

In particular, the emphasis is given to the specification, design and implementation of the gateway. In our model, the gateway is specified with a finite state machine. Based on the model, the gateway is designed by considering the issues of naming, name assigning, buffering, session handling, multi-session managing and error processing. A prototype of the gateway is implemented on a PC/XT computer with a LAN adapter, an ISDN adapter and the gateway program coded in "C". Performance benchmarks are conducted in terms of two factors: Message size and background traffic. These factors may affect the performance of internetwork data transmission. Finally, test results on different network conditions are given.

Acknowledgements

I would like to thank my thesis supervisors Dr. N. D. Georganas and Dr. W. F. McGee for their help, encouragement and financial support over the course of my graduate study. I would like to particularly thank them for their many useful comments in the writing of the thesis.

I would like to thank my thesis examiners Dr. O. Yang and Dr. R. Kaye for their comments on my thesis.

Thanks also to Dr. L Orozco-Barabosa for his many suggestions on my work.

Mr. J. Teo of the University of Ottawa, Mr. P. Lee of Mitel and Mr. J. Mastronardi of BNR are also acknowledged for providing me with valuable information.

Also, I would like to thank Mr. W. Williams and P. Neusy for their help during the preparation of the thesis.

Acronyms

ACK	Acknowledgement
ANSI	American National Standards Institute
API	Application Program Interface
BIOS	Basic Input Output Stystem
BRA	Basic Rate Access
CATV	Cable Television
CCITT	the International Telegraph and Telephone Consultative Committee
CSMA/CD	Carrier Sense Multiple Access with Collision
DCE	Data Circuit-terminating Equipment
DDN	Defence Data Network
DOD	Department of Defence
DOS	Disk Operating System
DPA	Department of Defence Protocol Architecture
DQDB	Distributed Queue Dual Buses
DTE	Data Terminal Equipment
ECMA	European Computer Manufacturers Association
EIA	Electronics Industries Association
FCS	Frame Check Sequence
FDDI	Fiber Distributed Data Interface
Gbps	Gigabits per second
GW	Gateway
HDLC	High Level Data Link Control
HSLAN	High Speed Local Area Network
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISO	International Organisation for Standardization
kbps	kilobits per second
LAN	Local Area Network
LAPB	Balanced Link Access Procedure
LAPD	Link Access Procedure D-channel
LLC	Logical Link Control

LSN	Listen
M	Message
MAC	Media Access Control
MAN	Metroplitan Area Network
MAP	Manufacturing Automation Protocol
Mbps	Megabits per second
ms	milli second
NCB	Network Control Block
NetBIOS	Network Basic Input Output System
NT	Network Terminator
OSI	Open System Interconnection
PBX	Private Branch Exchange
PC	Personal Computer
PCTA	Personal Computer Terminal Adapter
PDN	Public Data Network
PDU	Protocol Data Unit
PRA	Primary Rate Access
PVC	Permanent Virtual Circuit
RAM	Random Access Memory
REC	Receive
SAP	Service Access Point
SAPI	Service Access Point Identifier
SCB	Session Control Block
SNA	System Network Architecture
STE	Signaling Terminal Equipment
SVC	Switched Virtual Circuit
TA	Terminal Adapter
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TE	Terminal Equipment
TOP	Technical and Office Protocol
TP	Transport Protocol
VC	Virtual Circuit
WAN	Wide Area Network
WS	Workstation
XNS	Xerox's Network System

List of Figures

Chapter 1

Introduction

1.1 Introduction

In offices and corporations, personal computers have made it easy for individuals to store, manipulate and display information. One person can have on his or her desk the computer power of the old mainframe. However, the world should not be isolated. Whether information is in the form of a text file, a database or a program, it will probably pass through the hands of dozens of people. The introduction of the communication networks makes it possible for all systems (including computers, printers and other electronic devices attached in the network) to communicate over a common transmission medium. Such networks as local area networks (LANs), wide area networks (WANs) and metropolitan area networks (MANs) have been developed and standardized. The transmission rates range from several kbps (kilobits per second) to hundred Mbps (megabits per second). The latest networks can even operate in Gbps (gigabits per second) [ENG88] with optical techniques. The networks enable the attached systems to share resources and to exchange data. Some networks can support integrated services including voice, data and video transmission.

In order to allow the user to have access to resources beyond those available in a single network, we wish to interconnect the networks together. For example, two LANs which are located far from each other can be interconnected through a third wide area network. However, it is impractical to consider merging them into a single network because of the many differences from network to network in terms of their characteristics, performance and economics. The practice of network interconnection, called *internetworking*, is quite active. Many approaches to internetworking have been proposed based on the different platforms of existing networks. Consequently, standards for internetworking have been issued, e.g.,

CCITT X.75, ISO IP and DOD IP. The major design issues considered for internetworking are: addressing, routing, segmentation and reassembling. This thesis will present a type of gateway which connects LAN and ISDN (Integrated Services Digital Network). The gateway is implemented over NetBIOS which is a Session-layer interface developed by IBM. It provides an alternative way for a user to access remote LAN resources through the ISDN packet-switched service.

1.2 Outline of the Thesis

The next section in this chapter will overview the aspects of computer networks such as OSI standards and other standards, local area networks and wide area networks. These are the foundations of the network interconnection since internetworking is implemented based on the existing networks. On the other hand, since internetworking is a kind of networking technique, the problems encountered in internetworking are similar to the ones in a single network. Therefore, it is necessary to mention the techniques which have been developed in the environment of a single network.

In Chapter 2, a review of network interconnections is presented. Different aspects of internetworking are considered, which include the general design considerations of internetworking in different environments and the concepts of internetworking in terms of the different OSI layers. Also, the existing standards of internetworking and their applicable environments are discussed.

In Chapter 3, we will first discuss and evaluate the existing network adapters for LANs and ISDN. The characteristics of the interconnected subnetworks should be taken into account when the gateway is designed and implemented. Since all the subnetworks connected by the gateways use NetBIOS – a session-layer interface, the characteristics of the NetBIOS should be considered as well. Based on this information, an application-layer gateway are proposed.

In Chapter 4, a formal specification for the gateway is developed using a finite state machine. Emphasis is made on the description of session establishment, data transfer and session termination.

In Chapter 5, the design and the implementation of the gateway program are presented. The hardware and software configuration of the gateway system is considered as well. The main issues of the real time program consist of name assignment, session handling, buffer handling , data transmission and error recovery.

In Chapter 6, an end-user program is developed to benchmark the gateway functions with several different network conditions. The factors that influence the performance are studied. Finally, experimental results are presented.

In Chapter 7, concluding remarks are given to evaluate this gateway implementation. Also suggestions for further works are given.

Finally, appendices including the gateway program and the test program written in "C" are included for reference.

1.3 Overview of the Communication Networks

The concepts of computer networking imply that entities in different systems need to communicate. The 'entity' and 'system' used here are in a very general sense. Examples of entities are user application programs, file transfer packages, database management systems and terminals. Examples of systems are users, stations, autonomous computers and their associated software, peripherals, terminals and remote sensors. In some cases the entity and the system in which it resides refer to the same physical devices (e.g. terminals). In general, an *entity* is anything capable of sending or receiving information, and a *system* is a physically distinct object that contains one or more entities.

For two entities to successfully communicate, what is communicated, how it is communicated and when it is communicated must conform to some mutually acceptable set of conventions between the two entities involved. The set of conventions is referred to as a *protocol*, which may be defined as a set of rules governing the exchange of data between two entities. The key elements of a protocol are:

- *Syntax*: includes data format, coding, and signal levels.
- *Semantics*: includes control information for coordination and error handling.
- *Timing*: includes speed matching and sequencing.

Some important characteristics of a protocol should be considered. These characteristics are common issues in computer and data communications [STAL88]:

- *Direct* communication between two systems, if a point-to-point or broadcast network is used. *Indirect* communication, if systems connect through a switched communication network or several networks.

- *Monolithic* protocols: a single protocol is used for the communication between two systems. *Structured* protocols exhibit a hierarchical or layered structure. Figure 1.1 suggests a structured set of protocols, and shows the most extreme case of two stations connected via multiple switched networks.

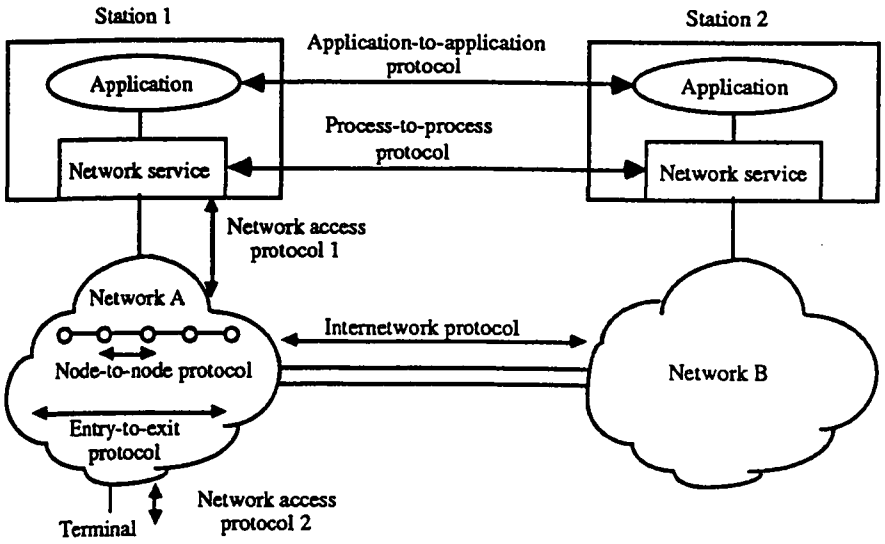


Figure 1.1: Relationship among communication protocols

- *Symmetric* protocols involve communication between peer entities. *Asymmetric* protocols may be dictated by the logic of an exchange, e.g., a 'user' and a 'server' process, or by the desire to keep one of the entities or systems as simple as possible.

- A *standard* protocol is the one shared by all systems. A *nonstandard* protocol is the one built for a specific communication situation or a particular model of a computer.

In general, a protocol includes the following functions:

- *Segmentation and reassembly.* If an application entity sends data in a continuous stream or in a large data block, lower level protocols may need to break up the data into blocks of some smaller bounded size, which has a number of advantages [STAL88]. A block of data exchanged between two entities via a protocol is referred to as a *protocol data unit* (PDU).

- *Encapsulation.* A data block is added with control information which normally contains address, error detecting code and protocol control information.

- *Connection control.* Data transmission can be *connectionless*, in which case data are transmitted in an unplanned fashion and without coordination, or it can be *connection-oriented*, in which case a logical connection should be established between the entities before data transmission. A well-known three-phase procedure for connection-oriented data transfer is:

  1. Connection establishment.
  2. Data transfer.
  3. Connection termination.

- *Flow control.* Generally, flow control is a function performed by a receiving entity to limit the amount or rate of data that is sent by a transmitting entity. The simplest form of a flow control is the *stop-and-wait* procedure.

- *Error control.* Some methods are needed to guarantee the correct transmission of data. If data are lost or damaged, retransmission or error report to upper level should be activated. As with flow control, error control is a function that must be performed at various levels of protocol.

- *Synchronization.* Two communicating protocol entities should be simultaneously in a well-defined state, for example at initialization, checkpointing, or termination. This is termed synchronization.

- *Sequencing.* Sequencing serves the ordered delivery, flow control and error control. This function only makes sense in the context of connection-oriented data transfer.

- *Addressing.* Communication entities are able to identify each other before and when they communicate. To achieve that,

  – a name is needed to specify what an object is;
  – an address is needed to specify where it is;
  – a route is needed to specify how to get there.

- *Multiplexing.* The multiplexing can be used in different context.

  1. Within one level, multiple simultaneous connections are allowed by using the connection names.

2. From one level to another, mapping of connection is permitted. Upper multiplexing occurs when multiple higher-level connections share a single lower-level connection. Downward multiplexing occurs when a single higher-level connection is built on

- *Transmission services.* The additional services for a protocol might be priority assignment, security mechanisms and grade of service, etc..

It should be emphasized that those characteristics and functions of the protocol are important not only for the communication between systems in a single network, but also for the communication between systems which are located in different networks.

Communication networks can be divided into two categories based on their architectures and techniques used to transfer data [STAL88], [TANE88]:

1. **Switched communication networks** using point-to-point connection, which include *Circuit-switched networks, Message-switched networks, and Packet-switched networks.*

2. **Broadcast communication networks** using broadcast channels, which include *Packet radio networks, Satellite networks, and Local networks.*

Also, concerning the geographical coverage, the communication networks can be divided into another three categories [SCHW87], [MCCO88]: *Wide area networks (WANs), Local area networks (LANs) and Metropolitan area networks (MANs).* A MAN covers the range of 10-100km [SCHW87], but uses LAN technology. Therefore, MAN will be treated the same as a high speed LAN in our discussion.

Many types of networks exist and operate in different environments. We will focus on some of them, such as IEEE LANs, X.25 network, and ISDN network. Those networks may be used as the fundamental components in an internetworking environment.

1.3.1 OSI Reference Model

In the mid-1970's, the development and use of computer networks began to achieve considerable attention. The early successes of the ARPANET [ROBE70], the immediate commercial potential of packet switching, satellite and local network technology, and the declining cost of hardware made it apparent that computer networking was quickly becoming an important field. When communication is desired among the computers from different vendors or even the computers from different models of the same vendor, the software development effort can

be a nightmare without a standard. Different vendors use different data formats and data exchange conventions. Therefore, a special-purpose approach to communications software development is too costly to be acceptable. The only solution is that computer vendors adopt and implement a common set of conventions. For this to happen, a set of international, or at least national standards must be announced by appropriate organizations. Such standards are of benefit to both vendors and customers.

Among standards-making organizations, the International Organization for Standardizations (ISO) has been proposing the *Open Systems Interconnection* (OSI) standards. The term *open* was chosen to emphasize that by conforming to OSI standards, a system would be open to communication with any other system obeying the same standards anywhere in the world. The basis of OSI is a *reference model* for the coordination of standards development. In OSI, three levels of abstractions are explicitly recognized: architecture, service specifications, and protocol specifications [DAY83]. The attractiveness of the OSI approach is that it promises to solve the communication problems between different computers.

The OSI Reference Model ISO 7498 [ISO83] was developed as a basic framework to deal with the complexity of communicating systems and the required protocol standards. Existing standards will be placed in perspective within the overall Reference Model. The most important idea in OSI is that it does not prescribe the way systems themselves must be made, only the way they must communicate. It is clear that the rapid development of applications and network types (e.g., packet networks, local area networks, satellite networks, cable TV networks, etc.) should not be inhibited. Moreover, it has to be possible to operate any given application over any given real network. Further, it is a requirement to be able to interconnect real networks together and operate a given application over any combination of different real network technologies.

A widely accepted structuring technique is *layering*. ISO chooses a seven-layer basic architectural structure. Each layer has a specific job. It performs a related subset of the functions to provide a service to the layer above. In its turn, it makes use of services from the layer below. The OSI Reference Model has two definitions:

1. *The OSI Service Definition* specifies what each layer must be able to do, what options are in it, and how the service is provided. It defines the commands and responses that the layer must be able to understand and make when communicating with the layers above and below. One of the key benefits of the OSI seven-layer Model is that layers can use any technology to perform their service, as long as the layer's service meets

the OSI service definition.

A service is formally specified by a set of *primitives* available to a user or other entity to access the service. The service primitives are divided into four classes:

- *Request*: An entity wants the service to do some work.
- *Indication*: An entity is to be informed about an event.
- *Response*: An entity wants to respond to an event.
- *Confirm*: An entity is to be informed about its request.

2. *The OSI Protocol Definition.* Each layer has a protocol definition, which defines the way that the layer communicates with their peers, i.e., the equivalent layers in the opposite systems.

In terms of the OSI terminology, any layer is referred to as the (N)-layer, while its next lower and higher layers are referred to as the $(N-1)$-layer and the $(N+1)$-layer, respectively. The same notation is used to designate all concepts relating to layers, e.g., entities in the (N)-layer are termed (N)-entities. The relationship among systems, layers , services entities and protocols are shown in Figure 1.2 (a) and Figure 1.2 (b).

The basic principles of the layering are listed in the Reference Model [ISO83] and could be summarized as follows [KNOW87] [TANE88]:

1. A layer should be created whenever a different level of abstraction is needed.
2. Each layer should perform a well defined function.
3. The function of each layer should be chosen with an objective toward defining international standardized protocols.
4. The layer boundaries should be chosen to minimize the information flow across the interfaces.
5. The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.

The reason for applying these principles is that OSI must not only accommodate existing applications but also those foreseen in the near future, and those as yet unforeseen but which

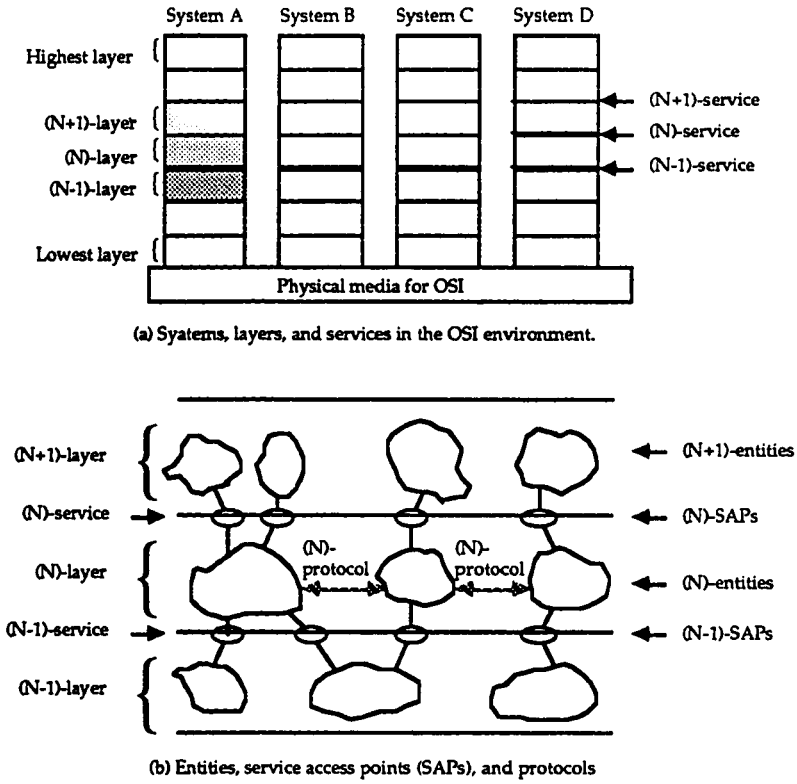


Figure 1.2: OSI layering

will inevitably be introduced well into the future. The two basic rationales of the layering are that:

1. Each layer obeys the layer above it, instructs the layer below it, and communicates as an equal with the equivalent layer on other systems.

2. Peer-to-peer protocols based on the services at any layer are independent of the protocols at any other layer. This permits changes to be made in the way a layer or a set of layers operate, provided they still offer the same service to the next higher layer.

The seven-layer structure of the OSI Reference Model is depicted in Figure 1.3. Each open system is logically composed of an ordered set of layers which together with the physical media provide a complete set of communication services. The following is a brief description of the layers from the reference [ISO83] that identifies some of the services provided by each layer:

- The *Application Layer* provides identification of intended communication partners (e.g.,

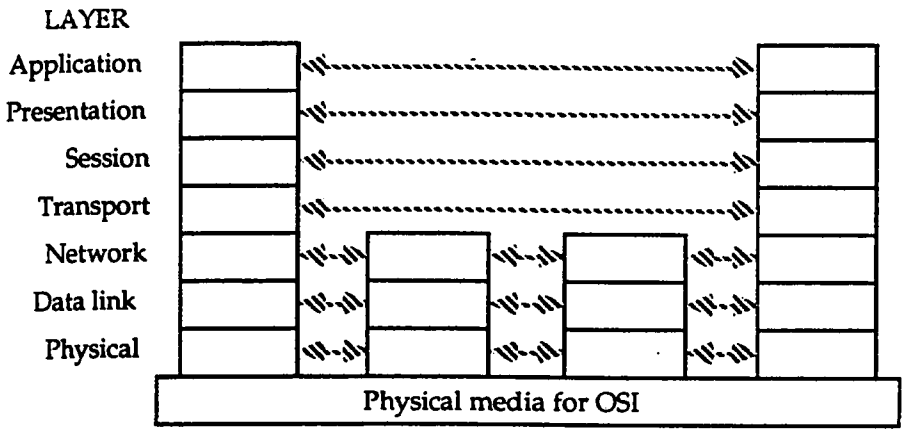


Figure 1.3: OSI Reference Model

by name, by address, by definite description) and identification of the subjects to be communicated (e.g., banking, text processing, airline reservations, determination of adequacy of resources, agreement on resources, and agreement on privacy mechanisms).

- The *Presentation Layer* provides data transfer and selection of the user data syntax.
- The *Session Layer* provides session connection establishment and release, turn management, session synchronization, and exception reporting.
- The *Transport Layer* provides end-to-end sequence control, flow control, error recovery, multiplexing, and blocking.
- The *Network Layer* provides quality of service, routing, sequencing, network flow control, and segmenting. It is the highest layer assigned to intermediate nodes in OSI Model (see Figure 1.3).
- The *Data Link Layer* provides error detection and correction, establishment and release of data link connections, link flow control, identification, and parameter exchange.
- The *Physical Layer* provides mechanical, electrical, functional, and procedural means to activate, maintain, and deactivate physical connections, and transparent transmission of bit streams.

Viewed as a system, the layers of the Reference Model can be broken into two groups. The bottom three layers, Physical, Data Link, and Network, cover the components of the network used to transmit the message. The top three layers, Session, Presentation, and Application,

however, generally reflect the characteristics of the communicating end systems. Their functions take place without regard for the physical medium actually used (see Figure 1.3). The Transport layer acts as the liaison between the end system and the network. Independence from the type, quality and number of sub-networks involved in communication between computer systems is inherent in the services offered by the transport layer [HENS88]. The unified view of OSI protocols can be found in [DAY83], [SCHW87], [KNOW87].

The ISO has traditionally been concerned with computer communications and distributed processing issues which correspond roughly to layer 4 through 7. Recently, this organization has worked forwards to the lower layers standards as well, such as in LANs. Another group involved in the international standardizations is the International Telegraph and Telephone Consultative Committee (CCITT). The CCITT has primarily been concerned with data transmission and communication network issues, which correspond roughly to the lower three layers of the OSI Model.

In terms of standards for data and computer communications, other groups are also involved, such as the American National Standards Institute (ANSI) in U.S. and the European Computer Manufacturers Association (ECMA), whose interests roughly parallel those of the ISO, the Electronics Industries Association (EIA) for standards of OSI layer 1, the Institute of Electrical and Electronics Engineers Inc. (IEEE) for standards of OSI layers 1 and 2 in local area networks.

The existing architectures, such as IBM's System Network Architecture (SNA) and communication networks, such as the ARPANET, performed most of the same functions as OSI but divide them differently. Figure 1.4 depicts the comparison of OSI Reference Model with the other architectures and other standards [VOEL86], where the DPA (Department of defense Protocol Architecture) is an outgrowth of the development of ARPANET and DDN (the Defense Data Network) [ELSA86].

1.3.2 Local Area Networks

Local area networks (LANs) have been developed toward widely accepted standards by IEEE and other organizations. They have mainly served as *factory automation* and *office automation*. Also, they are the major internetworking building blocks. The benefits of the LANs include: (1) sharing expensive computer resources such as large disk drives and high-quality printers, (2) exchanging information among members of a group, (3) interconnecting a set of devices (sensors, programmable controllers, and robots) to monitor and control

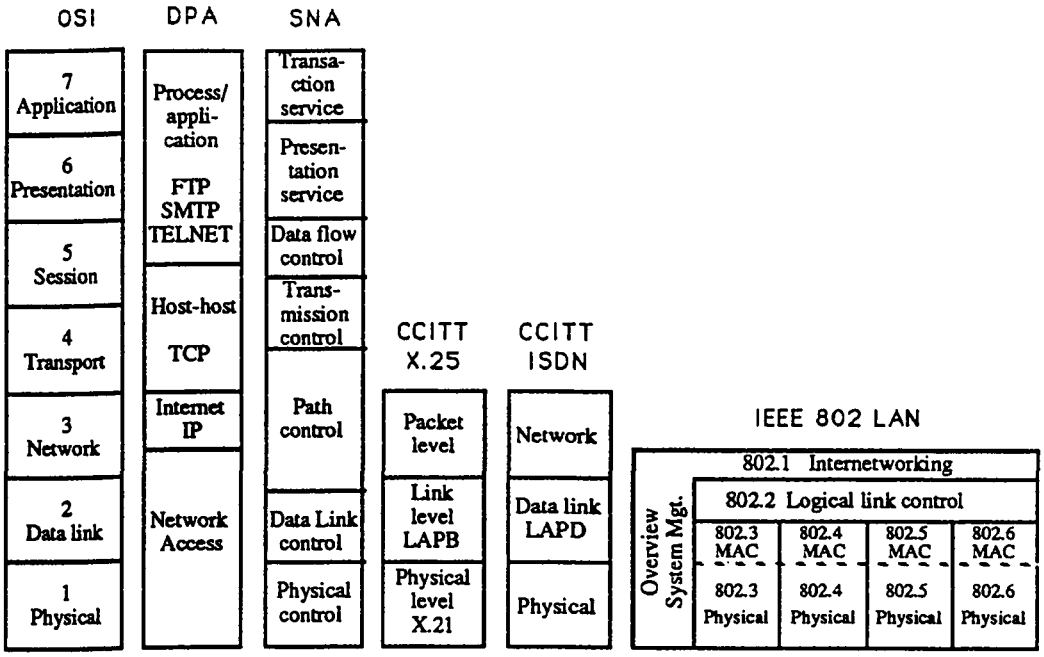


Figure 1.4: Comparison of OSI Model with other structures and standards

manufacturing operations, (4) interconnecting to other networks for common databases and mainframe access. It is obvious that a networked computer is much more powerful than a stand-along computer. With the expansion of the information exchange, both the connection of stand-along computers and the connection of networks become increasingly necessary and important.

LAN

Some of the typical characteristics of copper local networks can be summarized as [STAL87]:

- High data rates: from 1 Mbps to 20 Mbps,
- Short distances: maximum to 25 km,
- Low error rate: from 10^{-8} to 10^{-11}.

Since the local area network uses broadcast channels, referred to as random access channels, dynamic channel allocation protocols are required. Of those protocols, a set of standards have been developed by the IEEE and also adapted by ISO as international standards for the local area networks. They are the IEEE standard 802.3 for CSMA/CD bus, the IEEE standard 802.4 for token-passing bus, and the IEEE standard 802.5 for token-passing ring.

These standards cover the Physical layer and a part of the Data Link layer, referred to as MAC (Medium Access Control) sublayer. The IEEE 802.2 standard describes the upper part of the Data Link layer which uses the LLC (Logical Link Control) protocol and is common to the various MAC methods that are defined and supported by the 802 standards activities. (see Figure 1.4).

LLC provides the basic functions of:

- Initiation of control signal interchange,
- Organization of data flow,
- Interpretation of received command PDUs and generation of appropriate response PDUs,
- Error control and recovery functions in the LLC.

In addition to them, two types of data-link control operation are available:

- *LLC 1: Unacknowledged connectionless service.* It is a datagram-style service that allows the user to send and receive LLC frames without requiring acknowledgement.
- *LLC 2: Connection-oriented service.* It provides a virtual circuit form of connection between service access points. The result of this service is that sequencing, flow control, and error recovery can be provided for the Data Link layer.

IEEE 802.3: CSMA/CD The IEEE CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocol operates on a bus configuration. The algorithm of this protocol is quite simple. All stations listen for transmission on the line. A station that wishes to transmit does so only if it detects the channel is idle. Once a collision is detected during the transmission, the transmission is immediately aborted, and a jamming signal is transmitted to assure that all stations know that there has been a collision. After waiting a random amount of time, the station attempts to transmit again using CSMA.

In the bus topology, using CSMA/CD has some advantages such as simple implementation, fair access, good performance at low and medium load, easy installation of stations, no modem required, and wide usage. Some disadvantages are collision detection using the analog circuitry, minimum packet size required, poor performance under very heavy load, cable length limitation, nondeterministic behavior and no priorities.

IEEE 802.4: Token Bus The token-passing bus procedure is designed for manufacturing automation activities. The bus used in this method still acts as a broadcast simultaneous channel. All stations receive the same traffic at the same time if transmission delays are ignored. Therefore, a token must be addressed to the successor station. Only a station whose address is in the token can respond and utilize it. The physical location of the successor is not important; only its address is needed. All active stations must be assured of using bus. The token-passing bus must be circuited by a logical ring structure. Each station must remember the address of the station that passed it the token and the station to which it passes the token. The details of the protocol are complicated, which involves the issues of adding stations, deleting stations, fault management by token holder, ring initialization, and classes of service [STAL88].

The token bus can overcome some disadvantages of CSMA/CD. It supports the priorities, provides a guaranteed fraction of the bandwidth to high-priority traffic, and at high load, has excellent throughput and efficiency. However, the disadvantage is that the protocol is extremely complex and has substantially delay at low load. Its broadband physical medium needs modems and wide bandwidth amplifiers.

IEEE 802.5: Token Ring In a ring structure, there are several medium access techniques, such as register insertion, slotted ring and token ring. The token ring has become the most popular ring access technique and selected for standardization by the IEEE 802 committee. IBM's token ring network are compatible with the IEEE 802.5 standard [PITT87A] [IBM87]. It should be noted that a ring is not really a broadcast medium, but a collection of individual point-to-point links that happen to form a circle. A ring consists of ring stations and the transmission medium to which they are attached. The token ring operates with a token-passing procedure adapted for the characteristics of a ring. The data are transmitted in only one direction. A station that wishes to transmit must wait until it detects a free token passing by and changes that token into a busy one. The station then transfers data to the medium, where the data travel sequentially from station to station. Each station repeats the data, checking it for errors and possibly modifying some bits before sending it on. When the data return to the originating station, the station removes it from the ring and inserts a new free token on the ring.

The token ring has some advantages as of the token bus. Furthermore, it is full digital connection in the physical medium. A ring can be used with virtually any transmission medium, from twisted pair to fiber optics. The use of wire centers makes the token ring the

only LAN that can detect and eliminate cable failures automatically. The major disadvantage is the presence of a centralized monitor function which introduces a critical component. Another is that there is some delay at low load.

High Speed LAN and MAN

As the interest in multimedia applications increases, ever higher speeds and greater bandwidth must be made available. If a LAN operates over 50 Mbps, it is called high speed LAN (HSLAN). If a network operates over such a high speed and covers a large geographical areas it will be referred to as a Metropolitan Area Network (MAN). The typical MAN or HSLAN traffic is expected to include LAN interconnection, graphic and digital images, bulk data transfer, digitized voice, compressed digitized video, and conventional terminal traffic.

Because of the requirements for voice and video, an *isochronous transmission* has been added beside synchronous and asynchronous transmission. In the isochronous communications, the transmission of units (bytes, octets, etc.) are equally timed, thus providing evenly spaced transmission of bytes. The latter is important for voice and video because the evenly spaced transmission of bits allows the data communications network to operate much like a standard speech digitizer, thereby eliminating the need for buffering mechanisms. The MAN and HSLAN are mainly used as a backbone network for interconnecting LANs. Since MAN and HSLAN are only different from LANs in MAC sub- and Physical layers, it is easy to interconnect LANs, and therefore, to provide communications among LANs with less extra overheads.

Dozens of medium access protocols for HSLANs and MANs which are different from those of conventional LANs have been developed. Those high-speed channel access schemes could be classified into three classes as shown in Figure 1.5 [SKOV89]:

1. *Demand assignment access protocols*, which order message transmissions on the media by serving the attached stations in a cyclic or non-cyclic order.

2. *Fixed assignment access protocols*, in which the total network bandwidth is divided among the attached stations in the time, frequency, or code domain in a fixed manner. A station always occupies a part of the channel capacity, whether or not it has data to transmit.

3. *Adaptive assignment access protocols*, which are hybrids that combine random and controlled access features. At low loads, a station can transmit as soon as it senses

the channel is idle. Since data packet collisions may occur as the load rises, the adaptive protocols switch the current mode over to a more restricted conflict-free mode of operation, such as token-passing or TDMA (time division multiple access).

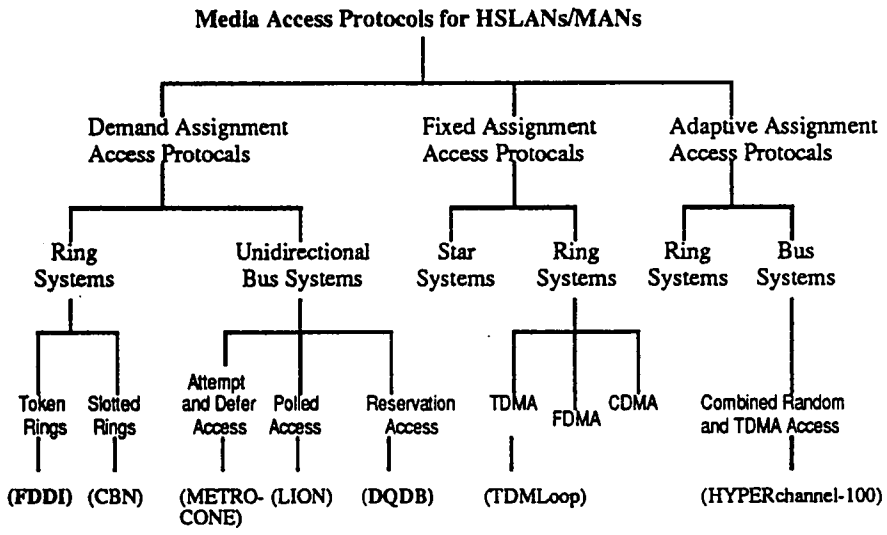


Figure 1.5: Media access protocols for HSLANs and MANs

Among of those MAC protocols, two of them are more popular, one is FDDI (Fiber Distributed Data Interface) which is a HSLAN standard in the United States [ANSI86], the other is DQDB (Distributed Queue Dual Buses) which is an IEEE MAN standard [IEEE88].

FDDI FDDI is a high performance fiber optic token ring network running at 100 Mbps over distances up to 200 km. FDDI-II is a modified FDDI to handle synchronous circuit switched PCM data for voice or ISDN traffic in addition to ordinary data [ROSS86], [JOSH86]. FDDI consists of two counterrotating fiber rings. The stations attached can be divided into two classes depending on the importance of station fault tolerance. Class A stations connect to both rings and the cheaper class B stations connect to one of the rings only. The FDDI standard employs the token ring protocol based on the IEEE 802.5 standard. However, many differences between them may be identified, such as in physical link, transmission rate, line code, clocking, token generation, reliability specification etc [STAL87]. The FDDI standards have accepted 802.2 (LLC) standards as the way to reach higher-layer protocols. Therefore, the FDDI MAC layer must also support the services required for communication at the interface between MAC and LLC sublayers.

Several levels of fault tolerance are possible in FDDI. For example, since wire centers can be used, if there is a cable fault in the cable going to a class B station, this station is cut off from the network, and the wire center provides a bypass for the node. If a cable fault happens in the cable going to a class A station, the two rings can be joined into a single ring. In FDDI, this reconfiguration happens automatically within a few milliseconds.

IEEE 802.6: DQDB IEEE 802.6 standard is a medium access control procedure for a MAN covering a large area such as a city or a campus. The physical configuration is a dual-bus architecture [MOLL88] [NEWM88]. Unlike FDDI, the structure of DQDB consists of a dual loop of the medium arranged physically as a ring but logically as a bus. The dual bus architecture carries unidirectional traffic in opposite directions. Each node is connected to both buses with read and unidirectional write capability. Thus, every node can communicate with every other node by sending information on one bus and receiving it on the other. One node, serving as master, generates the empty frames to two buses in different directions for the use of data transmission by the other nodes and does not repeat the data coming in from either direction. The distributed queueing MAC protocol is fundamentally different from existing LAN packet access protocols.

When failures occur in a node, no reconfiguration of the bus is necessary, because faulty nodes are easily bypassed. If the cable has to be cut or a node failure not be bypassed with relays, the logical opening can be closed. The original ends of the bus are instructed to pass data rather than terminate it. The detailed description of the DQDB protocol can be found in the IEEE 802.6 standard and [NEWM88].

1.3.3 Wide Area Networks

A wide area network (WAN) can serve a large area such as a city, a country, even an international zone. The use of many, or all, ISO layers is particularly important for WANs because of the complex topology (see Figure 1.6), wide geographical coverage, large number of nodes and links, extensive switching and routing, numerous points requiring flow control and congestion control, relatively long delays in end-to-end transmission, and long-distance communication requiring complex acknowledgement schemes. In contrast to WANs, LANs which are presented in the previous section have much simpler characteristics and thus have less need for the various ISO protocol services. For example, the LAN do not need the services normally provided by the Transport and Network layers, because routing, switching

and traditional flow and congestion control are unnecessary for them [SCHN83].

The WANs are important elements for internetworking. They are increasingly used as *backbone* facilities to interconnect dispersed sets of LANs. Certainly, a single system, such as a terminal, can be attached directly to the WAN as well. One such network is ARPANET. Among the types of WAN approaches are those that use satellites or leased lines, and the CCITT X.25 packet-switching specification for public data networks (PDNs) is the most widely used network access method. As a network protocol, it can be used in local (private) networks as well. The CCITT ISDN specification is another example of WAN which uses a new standard developed for public networks including the services of data, voice and video transmission [CCITT84].

Public Data network

Usually, the public data network is a mesh topology in which there are multiple communication links and relay points, referred to as switches or intermediate nodes (see Figure 1.6). Pairs of switches may be directly connected by a link or through an intermediate set of links and switches. Each pair of switches has many possible paths between them, creating a configuration with high availability. Therefore, a routing service is required for choosing a path between any pair of switches. In contrast, LANs broadcast packets to all systems attached in the network, rather than switching them between sender and receiver. The switches are functioning up to Network layer according to the OSI Reference Model.

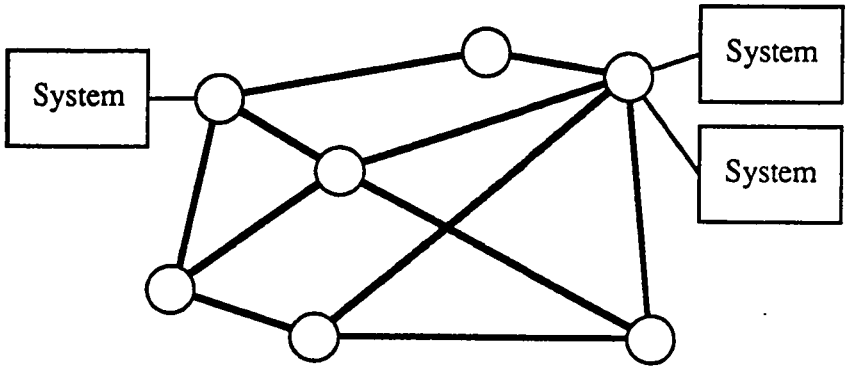


Figure 1.6: A WAN with mesh configuration

Systems are attached to certain points of the mesh through which traffic is exchanged with the mesh and thereby with other attached systems. The systems can be gateways,

computers, or special-purpose protocol converters for other devices.

A WAN with mesh configuration is most commonly a switched communication network. The communication may be through *circuit switching, message switching*, or *packet switching*. The packet switching will be focused on in this thesis. In the packet switched network, the data unit must be segmented into a set of packets and reassembled at the destination. A packet size is a fixed amount of user data and accompanying control information. The key elements of a packet switched network are:

- Routing: to route each packet through the network since the communication between systems has to be through a set of intermediate nodes.
- Flow control and Congestion control: to ensure that one user does not overwhelm the other user with packets and to maintain timely, efficient delivery of packets.
- Error control: to handle errors, to assure that the packets arrive correctly and in proper sequence.

X.25 The CCITT Recommendation X.25 is a standard device-independent *interface* between networks and user devices operating in the packet-mode [RYBC80] [SCHW87] [DEAS86]. The concept of the X.25 is depicted in Figure 1.7. There are two communication units defined, one is the data terminal equipment (DTE), the other is the data circuit-terminating equipment (DCE). The whole X.25 protocol is concerned with the interface between the DTE (the network user), and the DCE (the network itself). No mention is made of how the network should operate internally. The X.25 interface between the DTE and the DCE consists of three distinct levels of control procedures as shown in Figure 1.8.

1. *The Physical Level* specifies the use of a duplex, point-to-point synchronous circuit, thus providing a physical transmission path between the DTE and the network. CCITT X.21 is commonly adopted.
2. *The Frame Level*, referred to as Link level specifies the use of data link control procedures called LAPB (balanced link access procedures) which is compatible with HDLC (high-level data link control) standardized by ISO. Each message on the link is sent in a *frame*. The Frame level procedures define mechanisms by which those frames can be sent and received with the detection of errors created by the physical link, their correction by retransmission, and the overall control of data being sent over the link, i.e., the data (packet) in a frame is carried reliably between the DTE and the network.

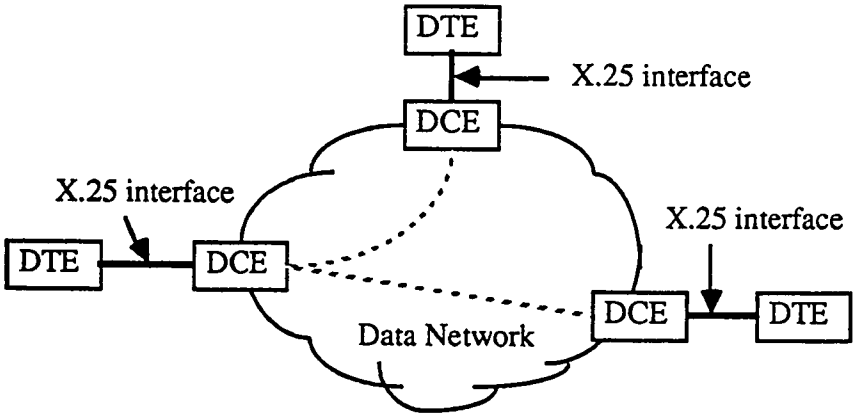


Figure 1.7: X.25 concept

3. *The Packet Level* specifies the manner in which control information and user data are structured into a packet. Its specification includes flow control, error control and the error handling procedures. This level is fully prescribed in X.25 specification. Three types of network services are available to X.25 DTE's. They are *virtual circuit* (VC) or called *switched virtual circuit* (SVC) or *virtual call*; *permanent virtual circuit* (PVC); and *fast select*. However, the fourth type of network services, *datagram*, has never been implemented and was dropped from the X.25 recommendation in the 1984 version.

   The virtual circuit service requires a virtual circuit to be established whenever data communication between end users. It is a temporary association between two DTEs and is initiated by one of the DTEs which signals a call request to the network. A permanent virtual circuit is a permanent association existing between two DTE's, which is analogous to a point-to-point private line. It requires no call set-up or call-clearing action by the DTE. In fast select service, the control packet that sets up the VC may carry short data as well.

Integrated Services Digital Network

The Integrated Services Digital Network (ISDN) can support a wide range of voice and non-voice applications including switched and non-switched in the same networks. It is basically a redesigned telephone system. A series of Recommendations by CCITT specify the concepts and principles of ISDN, its service capabilities, overall network aspects and functions, user-network interfaces, internetwork interfaces, and maintenance principles in

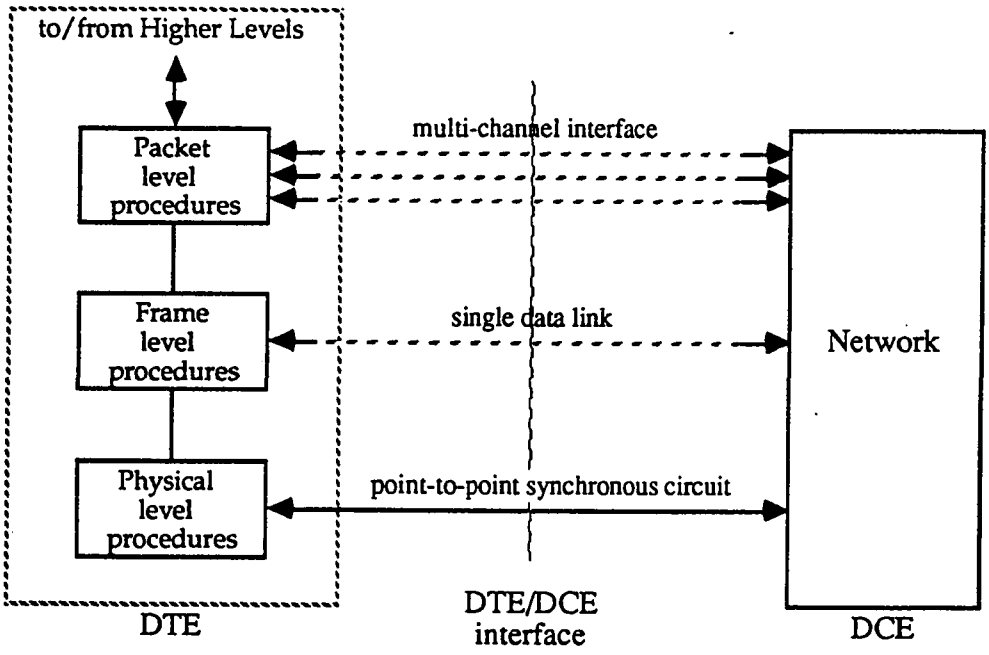


Figure 1.8: X.25 layers

the I-series recommendations [DECI86]. From the user point of view, ISDN is a type of WAN.

The ISDN user-network interfaces are defined by specifying these characteristics at well defined places in a network reference configuration. *Reference configurations* are conceptual (virtual) models useful in identifying possible physical access arrangements for connection to an ISDN. Two concepts are used in defining reference configurations:

- *Functional groupings* are sets of functions which may be provided in ISDN user access arrangements. Specific functions in a functional group may or may not be present and some functions may be performed in more than one functional grouping.

- *Reference points* are the conceptual points dividing functional groups. In some cases, reference points may correspond to a physical interface between two pieces of equipments. In other cases, no physical interface may exist at a reference point.

A brief outline of the functional groups shown in Figure 1.9 is as follows according to the CCITT Recommendation I.411 [CCITT84]:

- NT1 (Network Termination 1) is responsible for layer 1 (Physical) functions, such as

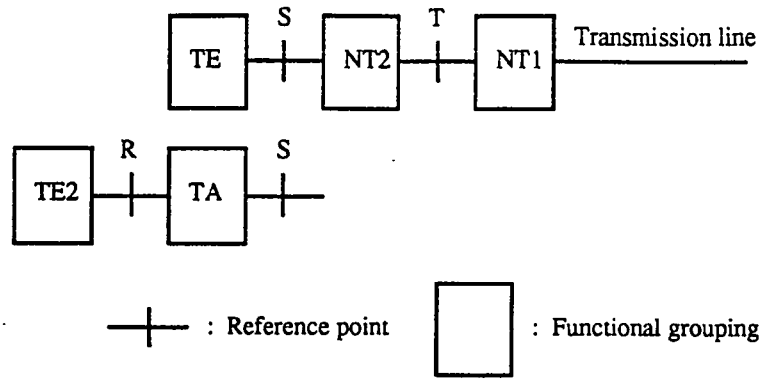


Figure 1.9: ISDN reference points and functional groupings

the line transmission termination, together with associated maintenance and line power extraction functions.

- NT2 (Network Termination 2) provides layer 2 and 3 (Link and Network) functions, allowing a variety of equipment to interface to the network.
- TA (Terminal Adapter) converts from the ISDN signaling protocols and channel rates to those of earlier CCITT standard interfaces such as X.21 and X.25.
- TE (Terminal Equipment) provides application and user interface functions to the customer.

The definitions of reference points T, S and R are given in the Recommendation I.210 [CCITT84].

In accordance with CCITT Recommendation I.412, communication across an ISDN is accomplished through channels. A channel is defined as representing a specified portion of the information-carrying capability of an interface, and can be treated as a bit-pipe. The two most commonly used channels will be the B channel and D channel. Others are H channels.

A B channel is a 64 kbps channel and is intended to carry a wide range of user information streams. The protocols used on B channel are shown in Figure 1.10. No signaling (or control) information for circuit switching will be conveyed across this channel.

A D channel is primarily intended to carry packetized signaling information for circuit switching by the ISDN and may have different bit rates depending on the type of access (either 16 kbps or 64 kbps). The D channel can also carry supplementary information such as packet data and telemetry which is transmitted along with signaling information using statistical multiplexing. In Recommendations, A link layer protocol of the D channel is

specified, called LAPD (link access protocol - D channel) which is based on HDLC and is close to the LAPB in X.25. At the Network layer, protocol I.451 is employed for control signaling.

H channels carry circuit-mode or packet-mode user information such as voice, data, video/image, and user-multiplexed information streams. The transmission rates are 384 kbps (H0 channel), or 1536 kbps (H11 channel), or 1920 kbps (H12 channel).

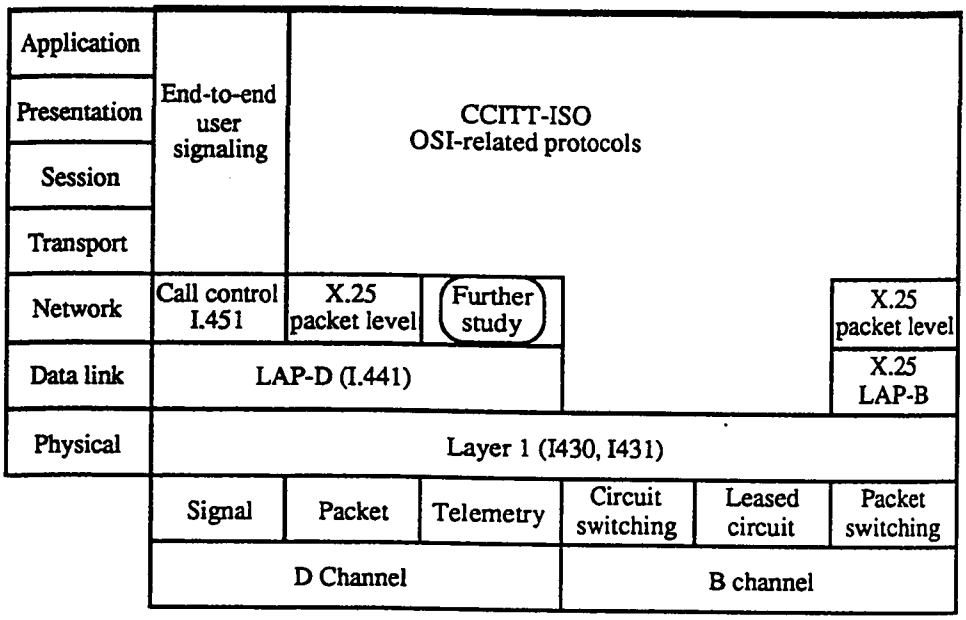


Figure 1.10: Protocol structure at the ISDN user-network interface

Channels are combined into interface structures which represent the maximum information carrying capability of a physical interface. These are defined in Recommendation I.412.

- Basic access interface structure: $2B + D$, where B channel is 64 kbps, and D channel is 16 kbps.

- Primary rate access interface structure (where all B and D channels are 64 kbps) :
  At the 1544 kbps [1]: $23B + D$, or $3H0 + D$, or $4H0$.
  At the 2048 kbps [2]: $30B + D$, or $5H0 + D$.
  Others: $H11$, $H12 + D$.
  Mixtures: $3H0 + 6B$, $2H0 + 11B + D$, etc.

[1] 1.544 Mbit/s North American Digital Trunk Standard–T1 Trunk
[2] 2.048 Mbit/s European Digital Trank Standard–CEPT Trunk

The provision of several B channels (i.e. two B channels in the basic rate access) provides an end user with several dedicated channels for voice/non-voice communication. These channels may be used to support independent calls at the same time.

The ISDN mentioned above is a narrowband one. A broadband ISDN using a packet oriented transfer mode and an asynchronous time division multiplexing technique is being introduced. It is a next generation of ISDN.

1.4 Summary

In this chapter, the characteristics of the communication networks and the OSI Reference Model are mentioned first. Although some structures for communication networks have existed before the OSI Model, the OSI Model became a widely accepted structure for communication networks, and also played an important role in the network interconnection which will be presented in the later chapters.

Some existing networks are then reviewed in this chapter, such as IEEE 802 LANs, FDDI, X.25 network and ISDN since they are not only working as a single network but also can be connected together to act as a large network from the user point of view. The networks that are interconnected may be LANs, MANs or WANs. It is necessary to understand the characteristics of the individual existing networks when we consider interconnecting them. For example, the significant differences between WAN and LAN must be addressed when planning and designing networks consisting of one or more LANs interconnected by one or more WANs. In the next chapter, the methods of the network interconnection are presented.

Chapter 2

Interconnection of Networks

2.1 Motivation

As mentioned in the previous chapter, LANs are widely used in data communication because of their performance capabilities, such as high bandwidth, low delay, high reliability and security. With the development of the technology and the increase in demand, what is needed is the ability to interconnect LANs so that any of the stations on any of the constituent networks can communicate. In a broad sense, interconnection of various networks (including LANs) is commonly termed *Internetworking*. There are many reasons for interconnecting preexisting networks, for example:

- A single LAN is limited to transmission ranges within a short distance.
- The resources of a single network are often inadequate to meet users' needs.
- The resources of a network may be shared by the users in other networks.
- It is impractical to consider merging the networks in a single network because so many differences exist in the different networks.
- A single network can become so large that it exceeds architectural limits on, for example, address space, or becomes unwieldy to manage in other respects.
- Machines embedded in the networks of different enterprises may need to talk to each other.
- Partitioning according to geography may be required to save on line costs.

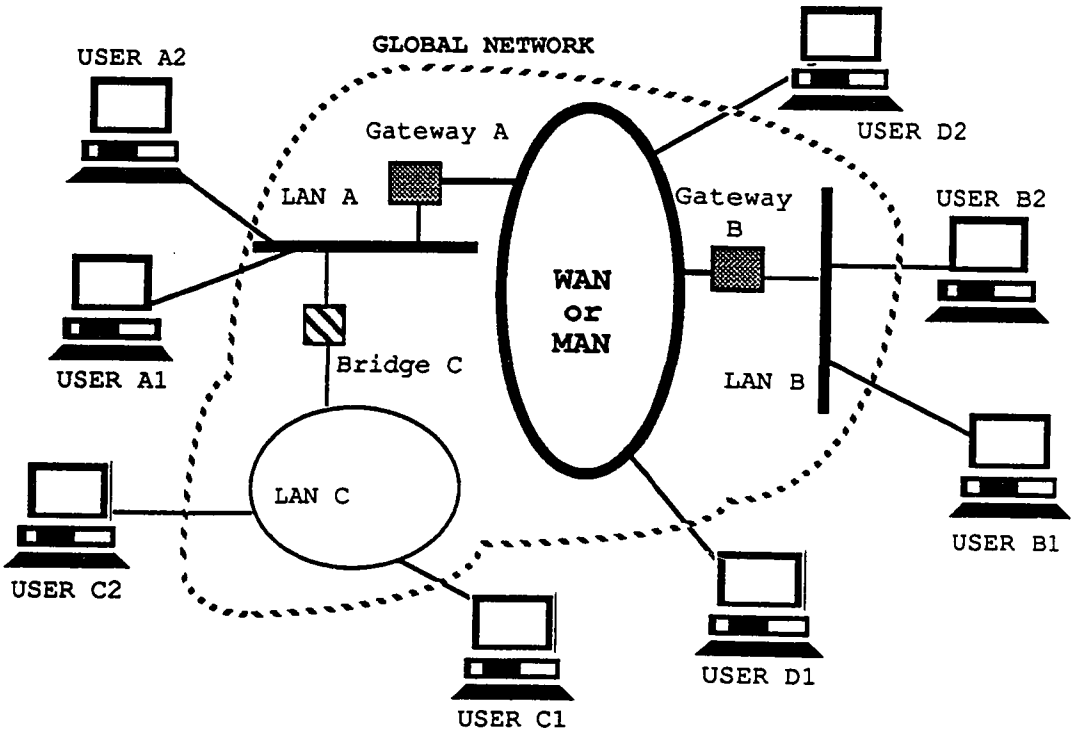


Figure 2.1: Internetwork

- Partitioning may be dictated by organizational subdivisions or by cost accounting or access control considerations.
- It may be desirable to put all the machines of different size ranges in different networks (e.g., all the minis and PCs in one and all the mainframes in another).
- Analogously, it may be desirable to partition according to link speed (e.g., all the T1-connected machines in one network and all the voice grade modem-connected machines in another).

From the user's viewpoint, working on an interconnected set of networks should not be different from working on a single network. The internetworking just provides an enlarged population of users with enlarged services (see Figure 2.1). Users should not be bothered by internetworking problems.

From a technology point of view, the internetworking services should include the following requirements:

1. Providing a link between networks. At least, a physical and link control connection is needed.

2. Providing for routing and delivery of data between processes on different networks.

3. Providing an accounting service that keeps track of the use of the various networks and gateways and maintains status information.

4. Providing the services listed above in such a way as not to require modifications to the networking architecture of any of the constituent networks. The internetworking facilities must accommodate a number of differences among networks.

2.2 Internetworking Issues

We have learnt how to interconnect computers in a single network. It is also important to find how to solve the problems of internetworking. There exist many types of networks with special characteristics, for example, a network may or may not follow the OSI standard. The networks which are interconnected are different in many aspects, such as addressing schemes, maximum packet sizes, interfaces, transmission speed, time-out values, routing techniques, and reliability. Many studies have been done regarding these incompatibilities between the networks. A number of specific technical issues related to internetworking are mentioned in papers on internetworking [GIEN79],[CALL83], [MCCO88], [CERF78] and [TANE88]. Although most of the internetworking issues are mainly concerned with the Network and Transport layers, in principle, these issues are applicable to the interconnection of networks at any OSI layer.

Architecture principles

Assume that the term *gateway* is used to refer to an internetworking unit in general in this section. A set of techniques developed for a single computer network can also be used for interconnecting homogeneous and heterogeneous networks, namely: *cascading, wrapping, multiplexing, switching* and *layering* [GIEN79].

- *Cascading* of services by gateways. When two networks with layered structures are identical, or not identical but the logical interfaces in x and y in Figure 2.2 are mappable; a request arriving at interface x must be translated into an equivalent request to

forward at interface y and vice versa. This gateway only translates formats but does not add services. Services S_A and S_B cascaded by a gateway must be equivalent. And they are also equivalent to the service S provided by the interconnected network.

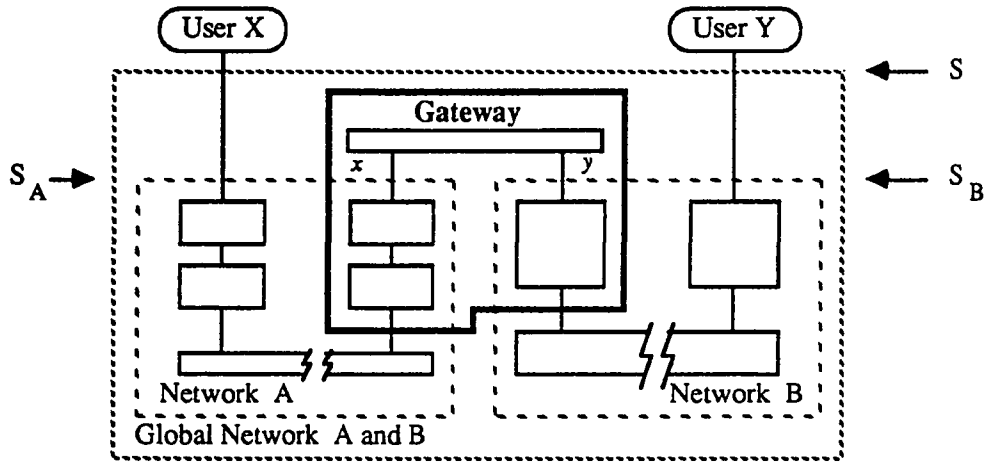


Figure 2.2: Cascading of services by gateways

- *Wrapping* of services in one of the underlying network(s). If there do not exist equivalent services in networks to be interconnected, it will be necessary either to select a common subset of both services which can be cascaded; i.e., to realize only a partial interconnection, or to add a new layer to one or both networks to reach equivalent services. Figure 2.3 shows the gateway applying this technique. The new layer appears only in user equipments and in the gateway, thus wrapping (end-to-end) the original network without requiring any modification to intermediate nodes (not shown in Figure 2.3) in the network.

- *Multiplexing* of gateways. A gateway can be shared among a number of internetwork activities by multiplexing the gateway among these activities. The interface between the gateway and each network can also be shared. If the network interface cannot be shared, the original network will have to be wrapped in an additional layer which will provide the interface multiplexing capability.

- *Switching* by gateways. Gateways usually have to decide to whom a given request should be forwarded, i.e., the gateways act as intermediate nodes in the interconnected network.

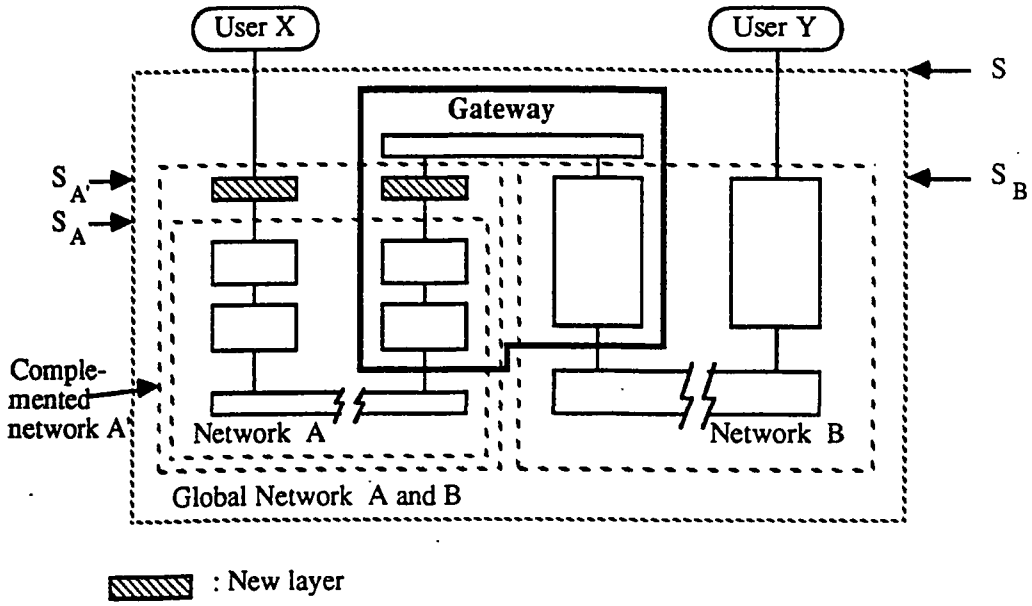


Figure 2.3: Wrapping of services in one network

- *Layering* of gateways. Gateways are also comprised of a layered structure. The highest layer at which the gateway performs its functions is referred to as the layer of internetworking.

Type of Service

Two types of services are used in internetworking:

- *Connection-oriented Internetworking.* The essential feature of this approach is that a sequence of virtual circuits is set up from the source, through one or more gateways, to the destination. Each gateway maintains tables telling which virtual circuits pass through it, where they are to be routed, and what the new virtual circuit number is. Once the virtual circuits are concatenated for a given source destination pair, all messages pass through the same gateways. This imposes some constraints with respect to reliability and efficiency.

- *Connectionless Internetworking*, or referred to as the datagram style. This is an alternative internetworking model to the connection-oriented one. The only service offered in this approach is the ability to inject datagrams into the subnetworks. all blocks of data, (e.g., packets in the Network layer), are not required in one connection to traverse

the same sequence of gateways. A major advantage of this approach to internetworking is that it can be used over subnetworks that do not use virtual circuits inside. This method is often used in many LANs and for LAN interconnections.

These two types of services are defined in [TANE88]. The OSI Reference Model and related service specification and protocol development have strongly emphasized connection-oriented service. However, there has been an increasing interest in connectionless services. It is becoming apparent that both services are complementary concepts and both are important for different environments and applications. Both approaches have different advantages and weaknesses in terms of delay, sequencing, and the simplicity of the implementation of a gateway etc. Properties of these two approaches for internetworking are discussed in [TANE88], [ROSS86].

Addressing

The main objective of internetworking is the extension of communication facilities to a larger number of users. Being able to address such a large population is therefore a key aspect of network interconnection. To accomplish the task of transmitting data from source to destination, each network needs to know to which destination data are to be delivered. At the lowest layers, it is typical to associate destinations with *addresses*. An address may be simply an integer or it may have a more internal structure. At the higher layers, however, it is more common to find text strings used as *names* of destinations. Typically, names are transformed into addresses which can be understood by the lower layers of the network. Although there is a clear definition of the concepts of name and address [DANT82], it is not necessary to distinguish them from each other. The actual address which a gateway must concern itself with depends on the layer that has been chosen for internetworking. A MAC layer gateway, for instance, has only a physical address to be considered. The objective of addressing is to ensure correct routing. Sometimes, multiple names and addresses for the same object (station) are permitted. There are several strategies of addressing in the internetworking environment [MCCO88], [DANT82] and [GIEN79]. Figure 2.4 depicts some of the addressing approaches.

1. *Unique address*, or referred to as *flat address*. The address of a station is treated the same in the subnetwork domain and in the internetwork domain (see Figure 2.4 (a)). In this addressing system, address ambiguity is resolved because there is no duplication in

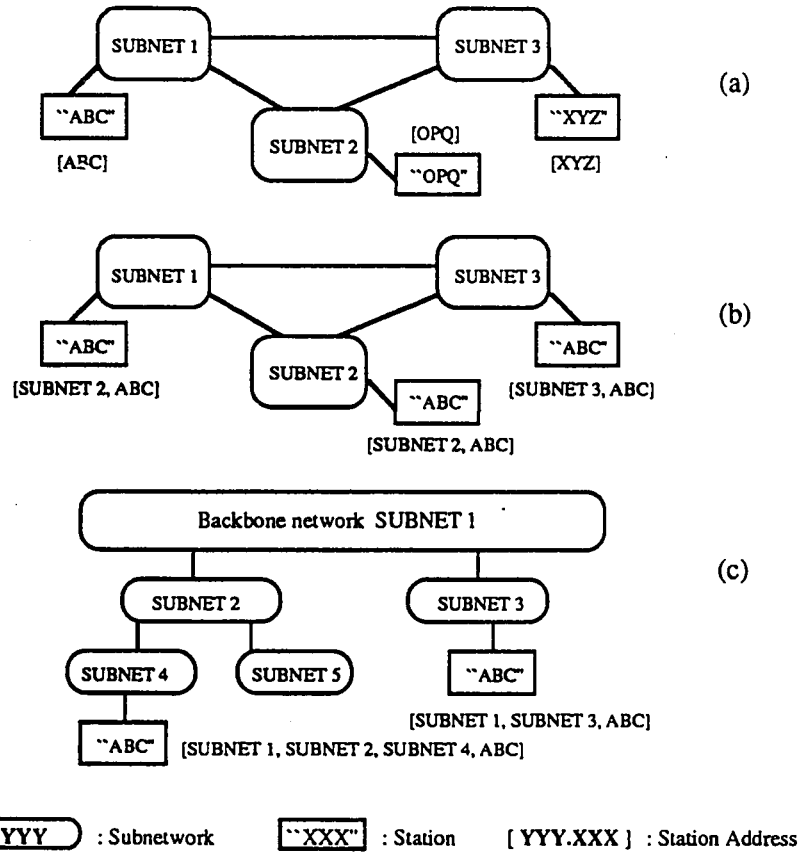


Figure 2.4: Examples of internetwork addressing

the whole interconnected network. It has the advantage of making stations independent of the subnetworks. It is not possible to derive from the address any information about the location. The whole length of the address has to be used whether a conversation happens within a subnetwork or between subnetworks. With the increase in the number of stations, the large address size would exceed the capabilities of available equipment. Therefore, this addressing approach is only suitable for small interconnected networks.

2. *Composite address.* Composite addressing adds information to the regular station address used in a subnetwork. The additional information allows each station to be identified within the internetwork environment while leaving each subnetwork free to assign and administer its own address space without being concerned with the addressing choices of other subnetworks (see Figure 2.4 (b)). One example of this addressing is to add a subnetwork address (or name) to a station address like address pair [subnetwork name, station name] [MCCO88]. Using the composite address, routing in a

gateway becomes simpler because the gateway is only responsible for delivering data to the destination subnetwork. Delivery to the destination station is left as a local matter for the destination subnetwork. In some literatures [GIEN79], this approach is merged into the hierarchical addressing approach.

3. *Hierarchical address.* If the subnetworks are hierarchical in nature (see Figure 2.4 (c)). Member subnetworks are nested inside others and are not visible until the proper higher-level subnetwork is reached. To reach stations within a hierarchical structure from the external environment, the actual destination station address may be hidden from internetworking since it is inside one or more subnetworks. A well known example of a hierarchical address system is given by the telephone system. Another example is the CCITT Recommendation X.121 address format for public data networks. This approach would also simplify routing. Individual subnetworks (particularly private networks) will in many cases be primarily used for local traffic, with only a very small proportion of internetworking traffic. It is desirable to use local addresses rather than full global addresses when conversations happen within a subnetwork. A local address in a subnetwork may be considered as a shortened full global address [ROSS86]. However, one limitation of the hierarchical structure of addresses is that it does not allow one station to be connected to more than on subnetwork with the same address [GIEN79].

Routing

Internetworking requires routing services associated with the processing of global addresses. A basic decision that must be made about routing services is whether they should be autonomous (*hop-by-hop routing*) or end-system driven (*source routing*) [DANT82],[MCCO88]. In hop-by-hop routing, the source specifies only the destination address and the routing decisions are taken by every gateway. In source routing [PITT87B], the source specifies all the intermediate routing decisions and includes this information along with the data being transmitted. Gateways do not have to maintain routing tables but only follow the routing instructions.

Besides the place where the routing decision is taken, routing is normally based on tables. The time and the way these tables are updated may widely vary. Two approaches can be classified. One is *fixed routing* in which the tables are set for a very long period based on estimated traffic. The other is *adaptive routing* in which the tables are periodically updated

to reflect changes of internetwork traffic [SCHW87], [DANT82].

Segmentation and Reassembly

Fragmentation and reassembly is required because many networks have a maximum allowable PDU size. For example, in the Network layer, the maximum PDU (i.e., packet) sizes in actual networks are different [TANE88]. Two basic approaches exist to solve the problem. They are:

- Intranetwork (transparent) segmentation and reassembly,which implies that an oversized data unit is segmented immediately before being passed to a subnetwork, and reassembled immediately after exiting the same subnetwork.
- Transnetwork (nontransparent) segmentation and reassembly,which implies that segmented data units are reassembled at the destination rather than any intermediate gateways.

For a reliable connection-oriented service, the problems with intranetwork segmentation and reassembly are much less severe. Thus, it is probably not necessary to use transnetwork segmentation. In a connectionless service, using intranetwork segmentation may suffer from the increase of buffering requirements at gateways and the increase of delays. Transnetwork segmentation and reassembly minimizes these problems by allowing gateways to transmit segments as soon as they are received[CALL83].

Flow Control

The object of internetwork flow control is to prevent the possibility of deadlock in gateways and to provide graceful performance degradation when gateways are overloaded with internetwork traffic. In addition, flow control is used to prevent network congestion. If an end-to-end flow control is applied, there is no flow control mechanism in the gateways.

Error Handling

Error handling in gateways may include error controlling and reporting. Error control functions are generally based on transmitting, back to the source station, control messages (positive or negative acknowledgements). In some networks, error control functions are performed

at various levels from the link control level (Data link layer) to the level of application-to-application communications over a network or a set of interconnected networks. Two kinds of error control services may be provided. One is to extend a cascade of error control within one network to the internetwork environment. The other is end-to-end error control over interconnected networks. In the second type of error protection, the gateway must forward acknowledgements transparently.

Some gateways may provide for some error reporting capabilities. For example, a reliable connection-oriented service will need to provide a *reset* capability to notify the user when there is possible loss, duplication, or missequencing of data, and an *abort* capability to notify the user when a connection must be closed due to an underlying error. Similarly, both connection-oriented and connectionless services may include the capability of notifying the user when a specified destination address is unknown or unreachable.

2.3 Approaches of Internetworking

ISO has defined, in the OSI Reference Model, seven layers within a network architecture. When two systems are not connected directly to each other, and connectivity is through an intermediary, the intermediary is known in ISO terminology as a "relay". There can be many relays on the path between two systems. If the relay shares a common layer n protocol with other systems, but does not participate in a layer $n+1$ protocol in the process of relaying information, it is known as a layer n relay.

Commonly implemented types of relays, together with commonly used terminology, are known as a *repeater* in the Physical layer, or a *bridge* in the Data link layer, or a *router* in the Network layer, or a *gateway* at any higher layer than the Network layer. The above names referred to the internetworking devices have been widely accepted for the discussion of network interconnections [PERL88], [STEW84], [STAL89]. The term "gateway", when used in networking literature [IEEE802.1], [MCCO88], [GIEN79], may refer to a Data link layer, a Network layer, or an upper layer relay, i.e., gateway denotes an internetworking unit in the general meaning.

2.3.1 Layer 1 Relay: Repeater

A *repeater* connects physical channel segments, effectively lengthening them (see Figure 2.5). The natures of repeaters depend on the physical channel technology. For example, in a

broadband cable system, the trunk amplifiers are also repeaters. Since the repeater is merely used to extend the length of the cable, this is not a true multiple network system. The repeater amplifies and retransmits all signals, including collisions, i.e., all traffic appears on all LAN segments. Therefore, the system behaves, from a practical point of view, as a single network.

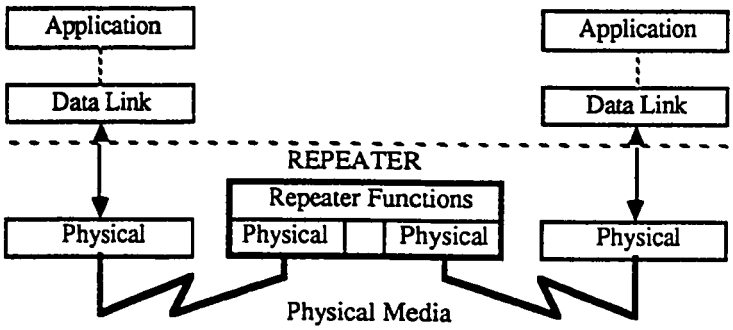


Figure 2.5: Repeater and physical channels

2.3.2 Layer 2 Relay: Bridge

A *bridge*, by definition, must operate totally within the Data Link layer and is not involved in the Network layer. Figure 2.6 shows the relationship of the bridge to the OSI model. The bridge which is a store-and-forward device should not require additional data link headers or actions by stations. It accepts an entire frame and passes it up to the Data Link layer where the checksum is verified, then the frame is sent down to the Physical layer for forwarding on a different subnetwork. Minor changes can be made to the frame before it is forwarded by the bridge, such as adding or deleting some fields from the frame header [TANE88].

Bridges are commonly used to interconnect IEEE 802 LANs. The IEEE 802 standards divide the Data link layer into Logical Link Control (LLC) and Media Access Control (MAC) sublayers (see Figure 1.4). IEEE 802 LANs use a common LLC protocol but different MAC protocols. Those bridges which are able to interconnect all IEEE 802 LANs are IEEE 802 bridges. They are referred to as MAC layer bridges because their frame forwarding actions do not involve LLC protocols [BERN85], [SEIF88].

Even though all LANs have the same characteristics, a bridged LAN is preferred rather than a single large LAN because of the following considerations:

- Reliability: It is dangerous to connect all data-processing devices in an organization

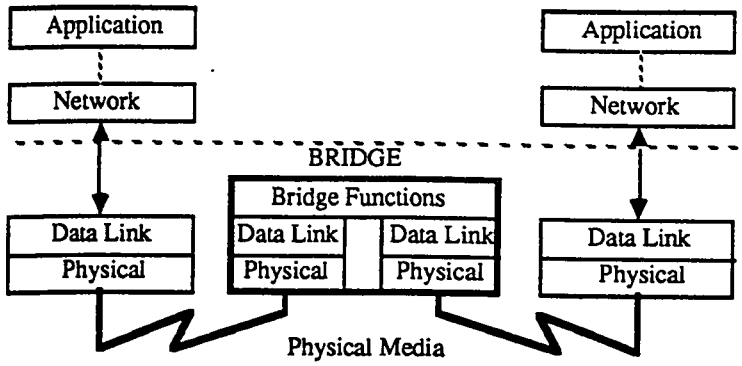


Figure 2.6: Bridge and the Data link layer

to one network because one fault in the network may disable communication for all devices. By using bridges, the network can be partitioned into self-contained units.

- Performance: A number of smaller LANs will often give improved performance if devices can be clustered so that intranetwork traffic significantly exceeds internetwork traffic.

- Security: The establishment of multiple LANs may improve security of communications. It is desirable to keep different types of traffic that have different security needs on physically separate media. At the same time, different users with different levels of security need to communicate through controlled mechanisms.

- Geography: Two separate LANs are needed to support devices clustered in two geographically distant locations.

Since there are several LANs interconnected, routing has to be introduced in the bridged LAN. Two of the many possible algorithms are standardized by the IEEE. One is the *Spanning Tree* [PERL88], [HAWE87], [LING87]. The bridge applying this algorithm is also called a transparent bridge. The other is the *Source Routing* [PITT87B], [PITT87C], and [BERN85].

The spanning tree bridge, referred to as transparent bridge, has a learn-and-decide scheme. To keep frames from looping, a loop-free topology is required. Therefore, bridges must first cut off an arbitrarily connected physical topology into a logical spanning tree. Suppose a bridge connects two LANs. The bridge maintains a database of the addresses of stations on each side. This database is empty at the beginning and watches all frames that

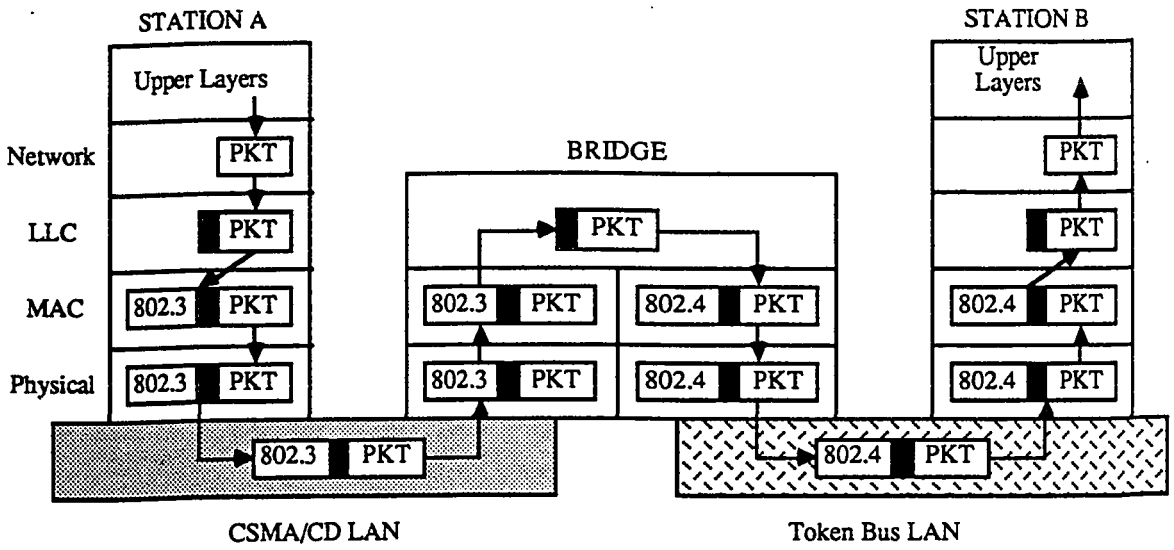


Figure 2.7: MAC sublayer Bridge

go by on the two LANs to learn the relative location of the stations. This information is put into the database. When a new frame is picked up by the bridge, it searches the database for the destination address contained in the frame. The forwarding of the frame to the other side is based on the results of this search. This simple algorithm requires no extra effort by end stations to switch frames through bridges, so that operation of the bridges is transparent. Since this type of bridge uses a *flat* address space, it is particularly well-suited to use with IEEE 802 LANs which exhibit consistent global address space administration[HAWE87].

Source routing bridge

Source routing is another routing approach. The term source routing stems from the specification by the transmitter (source) of the route that the frame is to follow. The concept can apply to any type of network, not just LANs [PITT87B]. The main issue with the source routing algorithm is how routes are discovered. A key element of dynamic discovery is the broadcasting of frames throughout the bridged LAN to explore all possible paths to the target. Broadcast frames are forwarded by bridges so that they travel all possible routes to all segments. A path is found when one or more of these frames reaches the destination station. When reaching the destination, all the broadcast frames containing route information will be returned to the source station along the recorded routes. The source can then choose the one to use, which will be placed in the header of each frame going to that destination.

This approach has to modify the original format of IEEE 802. For example, in a CSMA/CD frame a new routing information (RI) field is inserted [PITT87B]. To avoid duplicate copies forwarded by parallel bridges (i.e., more than one bridge between the same pair of LANs), the algorithm divides the original LAN number field into two parts - the LAN number and parallel bridge number - so that each route can be specified precisely. Since routes are chosen by source stations, the functions of the bridges are simplified.

Besides interconnecting LANs, bridges are capable of interconnecting networks whose functions in the Data Link layer are similar.

2.3.3 Layer 3 Relay: Router

Internetworking at the Network layer requires that any two Application layer users who actually wish to communicate must choose common protocols for layers 4 through 7, or must find a resource to do protocol translation. This does not necessarily require all users to use a common set of protocols. At the Network layer, relays are called routers as shown in Figure 2.8. Since the OSI Model assigns routing functions to the Network layer, the term router has come to be associated within this layer. A router stores and forwards packets directly addressed to it by end systems, with the expectation that the router will get information in the packet to the desired destination. The stations sending and receiving packets and the router must adhere to a common routing protocol. All routers require the active participation of the end users. Most of the internetworking issues which have been mentioned in the previous section have to be considered in the design of routers. There are two common routers: X.75 that uses a connection-oriented interconnection approach, and IP (Internet Protocol) that is in the connectionless strategy.

X.75

The X.75 standard was developed by the CCITT [CCITT80A] as a supplement to X.25. It is designed for interconnecting public X.25 networks. The CCITT has also established the International Numbering Plan for public data networks in the Recommendation X.121 [CCITT80B] which can be used as a standard to address users in an interconnected public network. X.75 defines procedures at the interface between two stations called Signaling Terminal Equipments (STEs) (see Figure 2.9). Similar to the CCITT Recommendation X.25, X.75 procedures are specified at three distinct level: (1) Physical level, (2) Link level, and (3) Packet level.

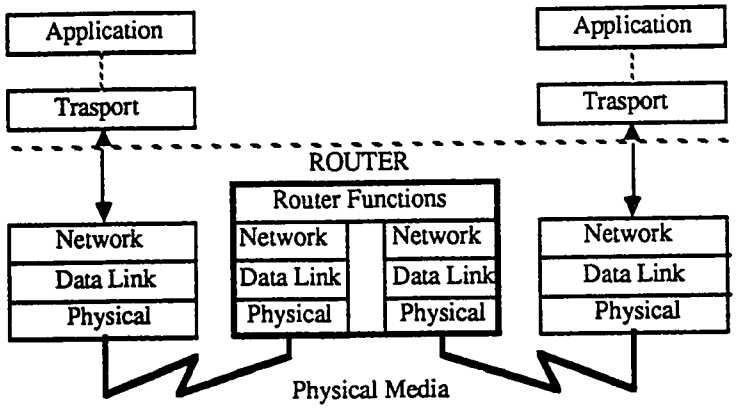


Figure 2.8: Router and the Network layer

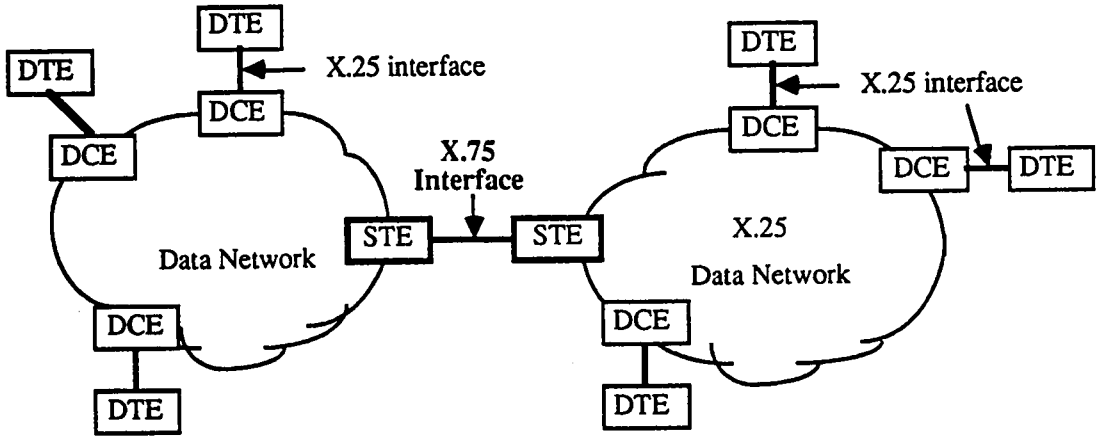


Figure 2.9: X.75

The Physical Level defines the electrical and functional characteristics of the interface between STEs. The Link Level specifies the use of the LAPB similar to the Link Level procedures of X.25. And the Packet Level defines procedures by which switched Virtual Calls (VCs) are established, maintained, and cleared across the X.75 interface (i.e. STEs). The specification of the Packet Level of X.75 is almost identical to that of X.25. The differences are those needed to accommodate internetwork administration and management functions of the STE. The other difference is that X.75 does not support the services of *permanent virtual circuits*. The internetworking operation of X.25 networks via X.75 provides a DTE-DTE virtual circuit that consists of a series of virtual circuits: DTE to STE (intranetwork), STE to STE (internetwork), STE to DTE (intranetwork). Each section is a distinct entity

with a separate virtual circuit, flow control, and error control. One of the X.75 application was discussed in [UNSO82].

The main attraction of using X.75 is the commonality of X.25. There are 186 national public data network identifier codes assigned by the CCITT [MCCO88]. Currently, there are more than a hundred X.25 public data networks, and many other private networks use this same protocol. Moreover, many LAN and mainframe vendors supply the X.25 packet protocol with their products. Choosing X.75 does not require the creation of another protocol layer. However, it does require that all subnetworks share X.25 packet-level protocols.

Internet Protocol (IP)

Two versions of a connectionless internetwork protocol in the Network layer exist. The Department of Defense (DoD) internet protocol (IP) [COME88], [STAL87C] has been in use since the mid-1970s, and ISO has adapted an internetwork protocol (IP) standard - ISO 8473, often referred to as ISO-IP later [MCCO88]. Both standards are very similar, although they differ slightly in protocol data unit structure [MCCO88]. The philosophy of IP is that gateways and stations share a common protocol for internetwork traffic but the subnetworks are otherwise undisturbed. In terms of the OSI model, IP fits in the Network (routing) layer. Therefore, the gateway using IP is a router. The IP address field has the form "net.subnet.host" [SEIF88], which is a hierarchical address. IP is simple and the router may be implemented in a small machine since IP has no provision for flow control or error control, and there are no acknowledgements of IP messages. IP only provides datagram service between stations. The service mechanisms of a reliable delivery are at a higher level. For example, the reliable delivery can be implemented by a Transport layer protocol within the end-to-end communication, such as TCP (Transmission Control Protocol) of the DoD [COME88] or TP 4 (Transport Protocol class 4) in OSI specification [SCHW87]. For example, The TCP uses a positive acknowledgement, time-out, and retransmission scheme to ensure delivery of all data. Each message is covered by an end-to-end checksum. It is obvious that the operation of the IP router is different from that of the X.25/X.75 approach. But the combination of TCP with IP will provide equivalent virtual circuit services to the X.25/X.75.

Figure 2.10 shows the operation of IP for data exchange between station A on LAN 1 and station B on LAN 2 through a backbone LAN. These stations and the routers must share a common IP. In addition, for successful communication, the two stations must share

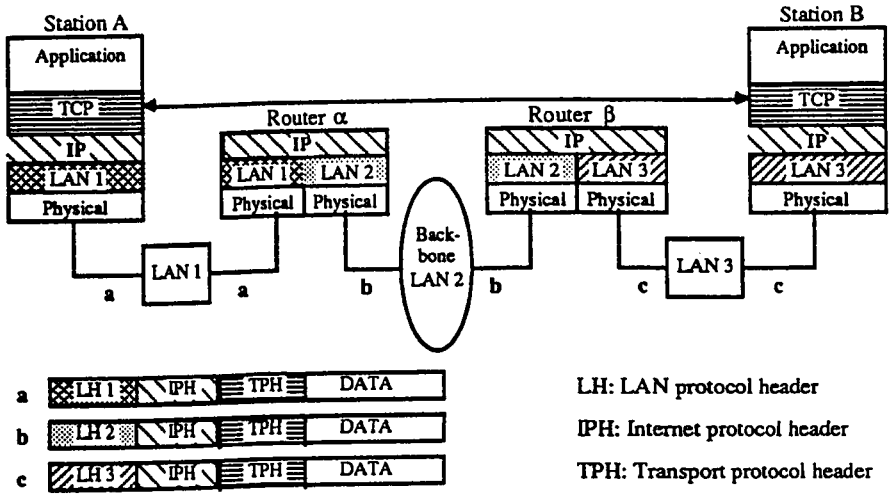


Figure 2.10: Internet Protocol

the same protocols above the IP. On the other hand, the IP makes no assumptions about the underlying network protocols. Almost any sort of network can participate, whether its internal operation is datagram or virtual circuit oriented. Each router must make a routing decision by analyzing the IP header. Also, the router may need to segment the datagram in order to accommodate a smaller maximum packet size on the outgoing network. This has been mentioned in the previous section in this chapter.

The IP approach means that each unit of data is passed from router to router in an attempt to get from source to destination. Since delivery is not guaranteed, there is no particular reliability requirement on any of the subnetworks. Thus, the protocol will work with any combination of subnetwork types. Since the sequence of delivery is not guaranteed either, successive data units can follow different paths through the interconnected network.

2.3.4 Upper layer relay: Gateway

Bridges and routers can be used to solve internetwork problems in an environment where all the devices implement compatible protocols up to the Network layer from the OSI model. However, there will be times when a proprietary network architecture such as IBM's SNA has already been installed. A gateway provides the way to permit the coexistence of OSI-based and proprietary products. Gateway functions at the Application layer tie together the dissimilar network architectures. Figure 2.11 depicts the relationship between a gateway and two dissimilar networks. Unlike the lower-level devices, gateways must be visible

to application processes, and explicitly communicate with them. Notice that Figure 2.11 shows reservations regarding how far down in protocol levels the gateway functions extend between the two network architectures. This is because there can be many types of gateways, interfacing with different layers in the network architectures. Therefore, the networks which are different on the upper four layers of OSI are interconnected by the gateway. Interconnection at a layer higher than layer 3 is particular interesting when interconnecting heterogeneous networks which use non-standardized protocols. Gateways are also used for the interconnection of homogeneous networks.

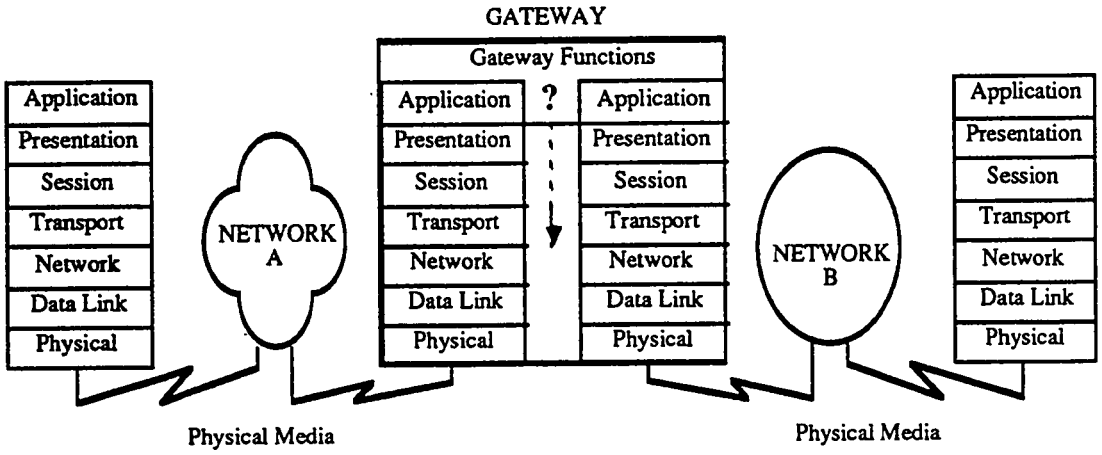


Figure 2.11: Gateway and the Application layer

Using gateways has several key limitations. For instance, the gateway is a potential bottleneck because more time has to be taken to process multi-layer protocols. If there is considerable traffic through the gateway, the performance becomes even worse. To overcome the performance limitations, more than one gateway might be needed. But this makes the station software complicated because the station software have to decide which gateway to use for each transfer.

Protocol converter An application protocol might be different in formats at each of the two network which are interconnected. The gateway must map or translate this protocol from one to another. One approach has been tried for accommodating the differences among networks; a special-purpose gateway, known as a *protocol converter*, can be built for each particular pair of dissimilar networks. The term protocol converter is widely used but loosely defined. A protocol converter is defined to perform a mapping between two protocols via their common protocol features. A number of approaches have been developed, such as mapping

two electronic mail systems [HU86], interconnecting SNA and XNS networks [ZOLI85], and interconnecting OSI and SNA networks [SY87].

2.4 LANs Interconnection

2.4.1 LAN-LAN

Direct Connection

Bridges are considered to be locally attached to the LANs they are interconnecting. Using these bridges, LANs can be concatenated to extend their range significantly. Transparent and source routing are two standards to interconnect IEEE 802 LANs.

Through Other Networks

The range of LANs could be extended still further through some wide area networks which can cover a large area. Wide area networks can be classified according to their operating speed, one is referred to as MAN, and has the transmission rate of several Mbps to Gbps, the other, referred to as WAN, is operated at several kbps.

LAN-WAN-LAN Figure 2.12 depicts a generic configuration for the interconnection LANs through a WAN.

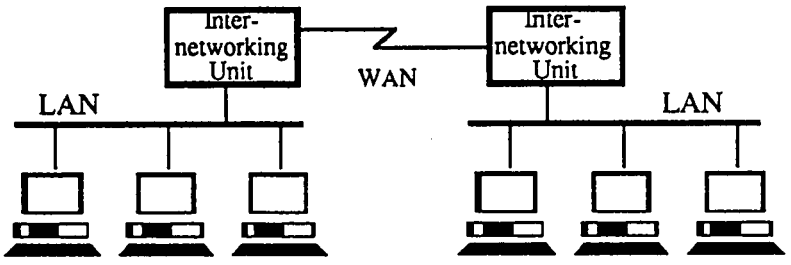


Figure 2.12: LANs interconnection through a WAN

The principle of this configuration is to keep the transmission as fast as possible. It is not necessary for a user on a LAN to talk with a user on the WAN. The conversion on the inter-networking unit could be simply implemented in the lower OSI layers. Some popular forms for such WANs are physical channels, frame relaying and X.25 packet-switching [LAMO89].

The WAN used to interconnect LANs can be physical channels. In this case, the internetworking units are referred to as bridges which convey MAC frames across those channels via layer 2 framing , such as HDLC (High Data Link Control) without performing the protocol procedure. These physical channels provided by a WAN can be either:

- Analog (voice-grade data) private lines, their data rates are usually limited to 9.6 kbps. Since the speed is too low relative to LAN speeds, it only can be used in a very light inter-traffic environment.
- Digital private lines, they are generally available at 56 kbps or T1 (1.544 Mbps in North America). The main issue for a user to select between those two rates may be the cost/utilization ratio since these lines can be expensive unless highly utilized.
- Digital circuit-switching includes ISDN and other circuit-switched networks. In the case of ISDN, it offers 2B+D Basic-Rate Access (BRA) and 23B+D Primary-Rate Access (PRA). Since each B channel can be circuit-switched independently to a different destination, a degree of multiplexing can be achieved over the access lines. See Figure 2.13.

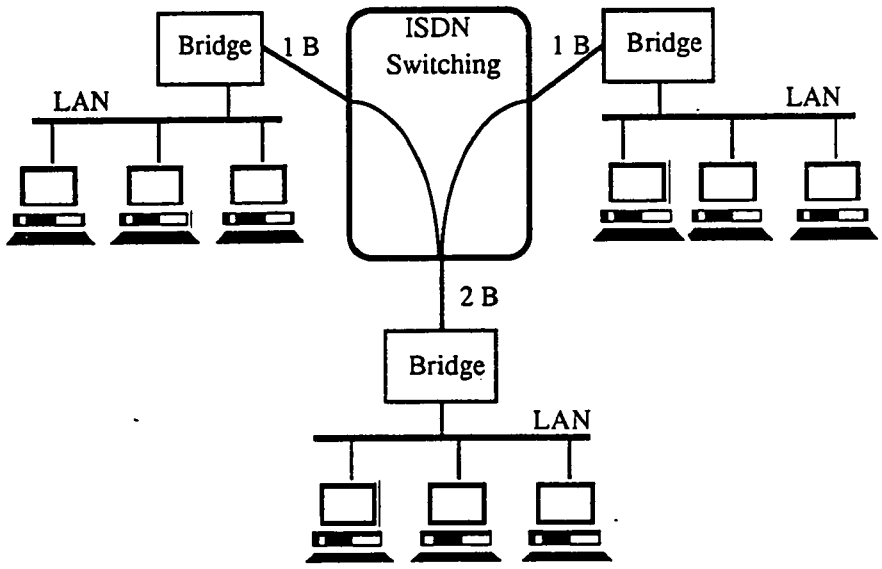


Figure 2.13: Multiplexing through ISDN Basic-Rate Access

The existing X.25 packet-switched WAN is a common carrier to link LANs. Since the link is established on a virtual circuit, the internetworking units are actually routers. With

the use of X.25, users can use more than one logical path over a physical link. However, speed is sacrificed because the routers have to perform the layer 3 protocol functions, such as retransmission of lost packets etc..

Frame relaying service is an ISDN additional packet-mode bearer service to be offered on both basic and primary rate interfaces. The basic premise behind frame relaying is the efficient transfer of LAPD (the CCITT Recommendation I.441) frames through an ISDN. Bridges, in this case, simply encapsulate the MAC frames into I.441 frames, and do not implement any I.441 elements of procedure. Such a structure to interconnect LANs was presented in the paper by W.S. Lai [LAI89] and by J. Lamont [LAMO89]. Figure 2.14 depicts an example of using the frame relay.

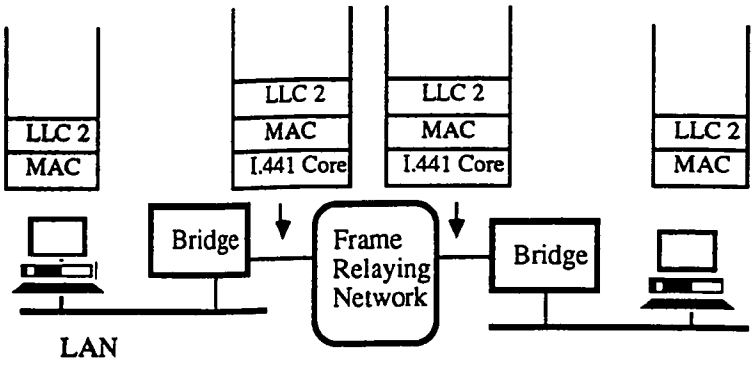


Figure 2.14: Frame Relaying

LAN-MAN-LAN Normally, MANs and HSLANs serve as backbone networks to interconnect LANs, for instance, using FDDI technology to extend the IEEE 802.3/Ethernet LAN [DONO90] and using DQDB technology to extend the IEEE 802.5 token ring LAN [VALL89]. To achieve the high throughput and reduce overhead, the internetworking units between LAN and MAN or between LAN and HSLAN are implemented as bridges in OSI layer 2. In the case of a LAN-FDDI-LAN configuration, a LAN frame is encapsulated in a FDDI frame (see Figure 2.15) [DONO90].

The other common realization of the LAN-MAN-LAN interconnection is adding IP layer both in LAN and MAN environments [DANT90].

When intermediate networks are MANs or HSLANs, the potential bottleneck will be the processing time of the internetworking units (bridges, or routers, or gateways). A multiprocessor approach would be applied to speed up the processing in the internetworking

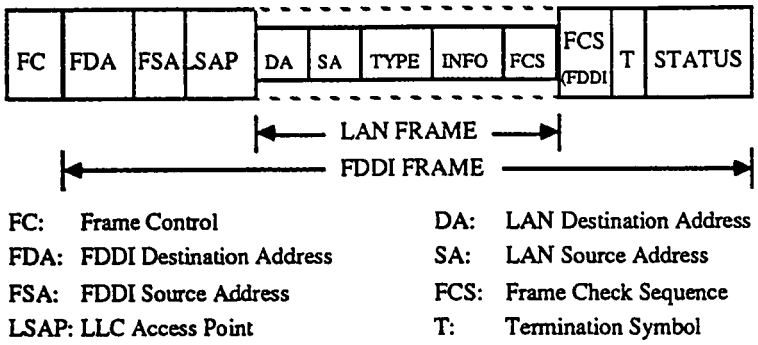


Figure 2.15: The encapsulation of a LAN frame in a FDDI frame

units [JENS90]. This approach distributes the concurrent tasks to several processors and to separates the receive and transmit processes into two processing units in order to increase the speed of data blocks passing through the internetworking units.

2.4.2 LAN-WAN

If a user on a LAN wants to communicate with a user on a WAN, a new common layer, for example, an IP layer, may be wrapped over both LAN and WAN. The IP layer can be implemented over the LLC layer of the LAN and over I.451 in the ISDN. The I.451 is the CCITT recommendation for the Network layer functions in ISDN D channel. I.451 signaling was initially based on X.25 signaling procedures which have been extended to accommodate for the wider range of services provided by the ISDN. Therefore, it is possible to map services from the LAN, in which the X.25 packet level protocol is implemented above the LLC layer, to the ISDN with the mapping technique[WILE].

2.5 Summary

The need for internetworking comes from the fact that there is not one network type that satisfies all computer-communications requirements. Rather, there are several technologies used. LANs provide high bandwidth over limited distances and are based on a passive media where data exchange is in a connectionless mode. WANs offer less bandwidth, are connection-oriented, and might base on circuit-switched or incorporate packet-switched data networks. Internetworking will make different networks work together to meet the requirements, such

as supporting more users, sharing large database resources and expanding the geographic dimension.

In this chapter, several issues were considered in the internetworking environment since networks perform the difference in many aspects. Then, the internetworking units, referred to as relays, are discussed in terms of the different OSI layers. Among the most commonly used relays are bridges in the Data Like layer, routers in the Network layer, and gateway in the layers above the Network layer. LAN interconnection is an important field in computer communication since there are many more LANs in operation than WANs. If the subnetworks which are interconnected are homogeneous, it is better to choose a lower layer relay to reduce the unnecessary overhead. If interconnecting different networks, a router or gateway has to be chosen to make the necessary protocol map or convert to others. In general, the methods of internetworking have to be varied according to the types of subnetworks and the protocols provided. In the next chapter, a gateway which interconnects a LAN and an ISDN will be considered. The gateway will be chosen over the Session layer for a couple of reasons which will be discussed in the next chapter.

Chapter 3

A Gateway on the Session Layer

In order to extend the services within a LAN to other LANs through ISDN, an internetworking unit is required to connect the LAN and the ISDN. In this chapter, we will first discuss and evaluate the existing network adapters for LANs and ISDN, and then select a suitable ISDN adapter among several ones provided by different vendors as the platform of the internetworking unit. Based on the hardware and software available, the layer of interconnection will be decided. Finally, the structure of the internetworking unit is developed to meet the requirements in our application environment.

3.1 Objective

Computer networking has made significant contribution to information exchange in practical applications. One of these implementations is a multimedia communications system for medical applications [GEOR89]. This system improves the communication of radiological information (images and diagnostic reports) between the radiologist and the client physicians in a hospital environment. The Sytek 6000 LAN has been used for the transmission of user data, digitized X-ray images and associated control data in this system.

Our objective is to expand the use of a resource out of one local area network environment via an ISDN link.

Since LANs and public networks are developed in different operating environments (refer to Chapter 2), it is impossible to merge them into a large network. Thus an internetworking device is used instead. Technically, what we should do in this thesis is to realize a class of internetworking that links the IEEE 802.3-based LANs through a narrow-band ISDN channel (B channel) for the application of data file transmission by means of the NetBIOS interface

(Figure 3.1). The interconnection of voice networks (PBXs) through ISDN is beyond our discussion in this thesis.

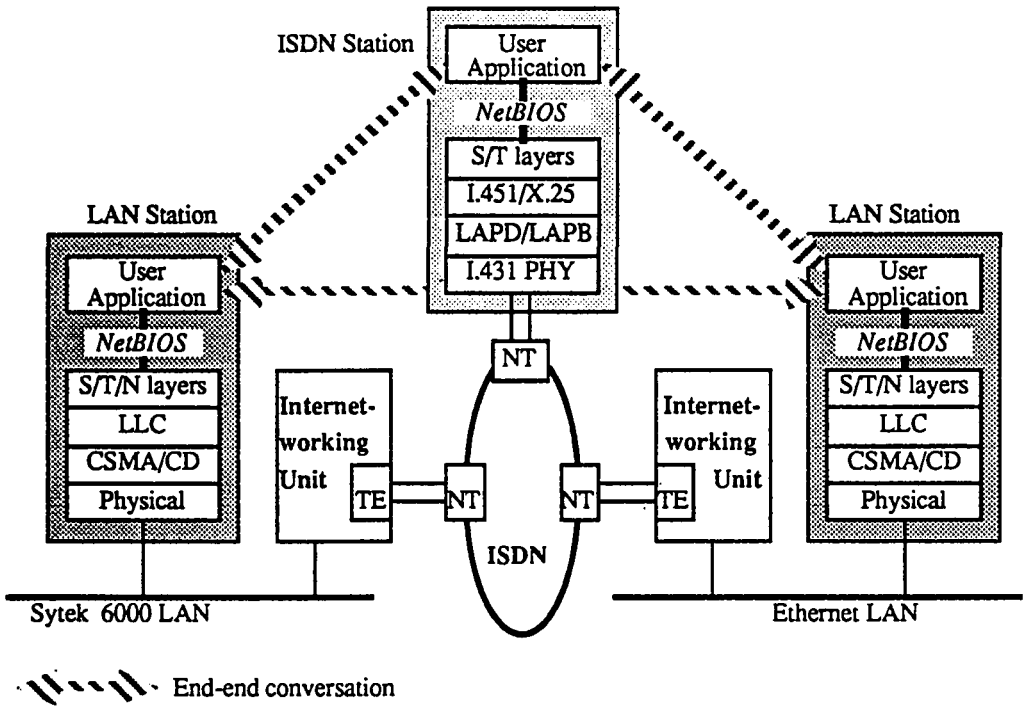


Figure 3.1: The interconnection of LANs through ISDN

The LANs which need to be interconnected are already in use and operate to support existing application software. It is expensive and, sometimes, not acceptable to change the communication protocols or to change their implementation in the LAN environment. In the ISDN environment, the CCITT specifies the D channel protocols up to the Network layer and the B channel protocols only in the Physical layer in the Recommendation [CCITT84]. It allows the user to define or choose upper layer protocols. It will simplify the LAN/ISDN internetworking, if a common set of services can be found in both the LAN and the ISDN. For the end-user, the existing applications should not be disturbed by internetworking (refer to Figure 3.1). That means the internetworking units should be transparent to all the existing applications.

There are many choices, which have been discussed in the previous chapter, for the network designer when networks are interconnected. For example, a bridge, a router or a gateway can be chosen as the internetworking device. And the internetworking service

can be either connection-oriented or connectionless. Consequently, LANs interconnection through ISDN may be realized with a bridge, such as the frame relay (in the Data Link layer)[LAI89], or the TCP/IP router (in the Network layer) [WILE],[LACH88],or other internetworking units in the high OSI layers. Among them, the simplest internetworking device is the bridge. In our particular internetworking design, however, we have to choose a proper internetworking layer in terms of the type of existing subnetworks, the availability of hardware and software interfaces, and the cost of implementation.

3.2 Subnetwork Environments

A single network (LAN or ISDN) is generally referred to as a subnetwork in internetworking. The subnetwork environments will affect the selection of the internetworking techniques. If two subnetworks are totally different, the internetworking will be the most complicated. The features and the services of the subnetworks should be considered before an internetworking approach is applied.

3.2.1 LAN: Sytek and Ethernet Adapters

IBM offers a variety of PC LAN adapters spanning five LAN environments: PC network broadband, PC network baseband, Ethernet, the IBM industrial network, and Token-ring [KLEE87]. Other vendors also provides similar network adapters. The LANs used as subnetworks for our prototype are the Sytek[1] System 6000 Network [SYTE86] which is compatible with the IBM[2] broadband PC LAN [IBM84A] and the Western Digitals baseband Thin Ethernet [WEST87].

The Sytek PC Network is designed to link together as many as 1,000 PCs within a maximum distance of 5 km [SYTE86]. This network provides a maximum 2-megabits burst rate on broadband cabling using the popular standard IEEE 802.3 CSMA/CD access method.

The Sytek 6120 PC Network Adapter card (compatible with the IBM PC Network Adapter) itself is a single add-on board inserted in the PC. It contains an Intel[3] 80188 microprocessor with 32k bytes of ROM (read-only memory) for on-board protocol processing, an 8k-byte ROM with the NetBIOS code, a ROM with the node identification, 16k bytes of RAM (random-access memory) for data buffers and table storage, Intel 82586 "Ethernet"

[1]Sytek is a trademark of Sytek Inc.
[2]IBM is a registered trademark of IBM Corporation
[3]Intel is a registered trademark of Intel Corporation

CSMA/CD chip operating at 2 Mbps for link level processing, and a complete CATV(cable television)-compatible RF (radio frequency) modem [SYTE86], [IBM84B]. The adapter uses the NetBIOS interface to communicate with the network and exploits this by offloading the local networking processing from the host CPU (Central Processing Unit) to the adapter processors [SYTE86].

The PC Network adapter supports the OSI functions up to the Session layer. The Physical layer of the adapter is implemented with a CATV-compatible RF modem. Employing standard broadband technology, each PC LAN Adapter transmits at 50.75 MHz and receives at 219 MHz. Therefore, a network translator (head-end) is required for frequency conversion. This technology ultimately allows the coexistence of local network and video signal on the same cable by FDM (frequency multiplexing modulation) techniques. The Data Link layer conforms to IEEE 802.3 (CSMA/CD) and provides a connectionless service. It is responsible for packet framing and CRC error control. This layer is implemented primarily with the Intel 82586 controller. The other three layers (Network, Transport and Session) are implemented using the Intel 80188 processor and ROM on the adapter. The routing based on address fields with in each packet is a function in the Network layer. Reliable point-to-point connections on the top of the connectionless service, data acknowledgements and flow control are the functions of the Transport layer. Communication sessions are established between stations using name pairs in the Session layer.

The Network layer is expandable in the PC Network adapter implementation in order to allow for future internetworking. Figure 3.2 depicts the multi-processor structure of a LAN station. This structure improves the host computer's performance, reduces its memory requirements, since the network code is internal to the adapter and not in the microcomputer's memory, and simplifies programming.

The EtherCard PLUS[4] [WEST87] is another IEEE 802.3 network adapter which has been used in our gateway implementation. This adapter allows one to set up a LAN using either Ethernet or Thin Ethernet configuration for network connection. It has an on-board RAM buffer of 8k bytes that speeds up network operations and eliminates the need for using DMA (direct memory access) channels. The buffer can work with the PC clock speeds from 4.77 to 16 MHz. Unlike the PC Network adapter, the operation of the NetBIOS software occupies the host CPU time in this card because this software is located in the PC RAM. The EtherCard PLUS adapter is designed for the PC, PC-XT, PC-AT and PS/2 Model 30 computers. By using this adapter, a Thin Ethernet network can support up to 100

[4]EtherCard PLUS is a trademark of Western Digital Corporation

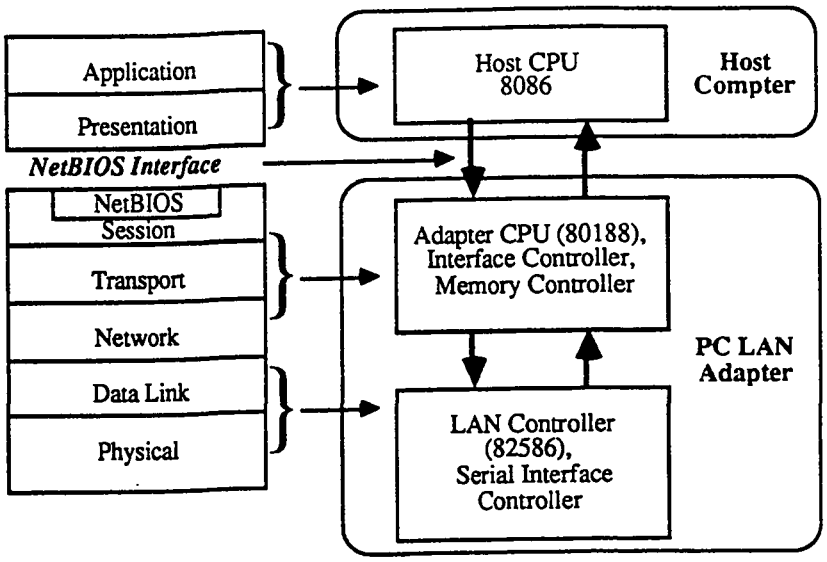


Figure 3.2: PC Network station relative the OSI layers

LAN stations along a 300-meter cable segment. The data rate on the cable is 10 Mbps. In terms of the OSI layering structure, the adapter functions the IEEE 802.3 (CSMA/CD) and is compatible with the higher layer software, such as ViaNet[5], Netware[6] and NetBIOS packages [WEST87].

3.2.2 ISDN: PCTA and Other Adapters

Northern Telecom's ISDN PCTA

The Northern Telecom IBM-PC based ISDN Personal Computer Terminal Adapter (PCTA) [NORT87A],[NORT87B],[NORT87C], [NORT87D],[NORT88] is a card which can be inserted in the peripheral expansion bus of the IBM PC, PC-XT, PC-AT, Northern Telecom M6000, and some compatibles. The PCTA attaches to the ISDN at the T reference point (refer to Figure 1.9) in the TE mode. It can be set in the NT mode for the network emulation. It also allows the PC to communicate with an ISDN T2317 digital telephone [NORT87E] on the same reference point. In the Figure 3.3 (a), the U reference point[7] is an interface between NT1 and ISDN network, which is a two-wire digital transmission standard being prepared

[5]ViaNet is a trademark of Western Digital Corporation
[6]NetWare is a trademark of Novell Incorporated
[7]The CCITT did not define the interface at the network side of the NT1

by the American National Standard Institute (ANSI T1.601-1988). With an application program–the Call Manager, the ISDN digital telephone T2317 can be controlled from the PC keyboard without touching the T2317.

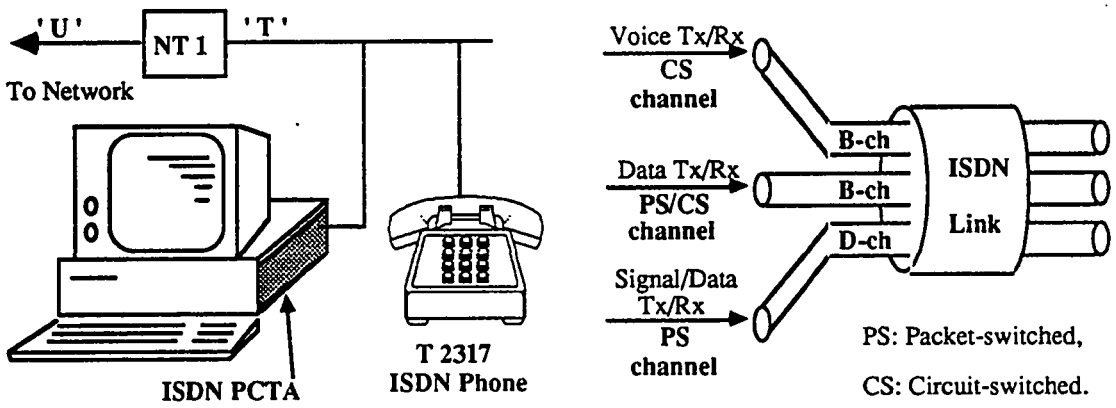


Figure 3.3: ISDN terminal adapter and ISDN phone

(a) connection (b) Data and voice in different channels

Figure 3.3 (b) depicts the services provided by the PCTA. Data can be transmitted on the D channel and one B channel in the packet-switched mode, or on one B channel in the circuit-switched mode. The other B channel is for the voice transmission. If the B1 channel is reserved for the digital telephone, the B2 channel is used for data transmission, or vice versa. Therefore, simultaneous accesses to one B channel (user selectable) and the D channel for data transmission are provided.

The PCTA system contains three main parts:

- *Terminal Adapter Hardware.* The PCTA's hardware consists of a full-length main card, which provides the PC interface and D/B channel control, with a full-length daughter board attached which provides support hardware, serial processing, and the ISDN T-reference-point interface. The new design board has integrated this two-board adapter into a half-length card with custom VLSI chips. The connection with the PC is in the I/O slots of the PC. And the connection with ISDN T interface is through a special eight-conductor line cord.

- *Terminal Adapter Firmware.* The firmware controls and manages the hardware. It consists of a bootstrap ROM, which provides immediate control of the PCTA after

powerup, and a program in RAM, which provides flexible software control of the PCTA's D channel and B channel functions and protocols. The D channel and B channel control firmware is stored on disk and is downloaded to the PCTA upon installation of the ISDN base software. The firmware provides control and signaling on the D channel according to the ISDN protocols. In the CCITT Recommendation for ISDN [CCITT84], the B channels could be handled with whatever protocols are necessary for the particular application at the time. In terms of the OSI model, the firmware performs the layer 2 functions of an ISDN PCTA.

- *Terminal Adapter Software.* This is PC-based software which is loaded from disk into the PC RAM upon request. It provides control over the PCTA's operations, management of logical data channels, and management of the PC's facilities, such as direct memory access for data transfer to the board. It also provides a software interface for all applications that involve the PCTA.

In summary, the ISDN PCTA support the following functions [NORT88]:

- ISDN layer 1 T interface (point-to-point, short, and extended passive buses);
- ISDN layer 2 Q.921 (I.441) multi-frame LAPD;
- ISDN layer 3 Q.931 (I.451)functional signaling protocol for circuit-switched calls;
- T-link protocol for circuit-switched calls (supports the asynchronous data rates of 1200, 2400, 4800 and 9600 bps);
- X.25 layer 2 LAPB;
- X.25 layer 3;
- NetBIOS interface at the Session Layer.

The combination of the MS-NET[8] and the ISDN PCTA can transform the applications in a LAN environment into a WAN environment. The capabilities of IBM's 3278 terminal emulation and VT100 terminal emulation through ISDN channel on standard telephone wiring are also available with the PCTA.

[8]MS is a trademark of Microsoft Corporation

Other ISDN Adapters Available

Beside the Northern Telecom ISDN PCTA, other vendors have also developed ISDN products. Two of these products will be evaluated here. They may be available as platforms for the LAN interconnection.

The Mitel ISDN Express Card [MITE87] was designed based on the Mitel ISDN custom chips by Mitel Semiconductor Corporation. The card itself acts as the Physical layer communication medium. It includes components for 2B+D Basic Access Rate at the S reference point (four-wire connection) and U reference point (two-wire connection), the Primary Access Rate reference points (T1 23B+D and CEPT 30B+D), a digital telephone, and switching and protocol functions. This card is connected to the PC in a PC I/O slot. The card provides all the narrow-band ISDN interfaces. Therefore, the rate of the ISDN link is selectable in terms of the actual traffic. The board, however, does not include the microprocessors. Therefore, all the functions from layer 2 to layer 7 have to be run in the host CPU. Low speed microcomputers, such as PC-XT may not have enough processing power to handle data at 64 kbps. In order to access a 64 kbps link freely, a high speed microcomputer is required.

To support the operation of the ISDN Express card, Mitel developed a set of software tools referred to as LAPD Toolkit package [MITE88]. This software operates in IBM PC family of personal computers under the MS-DOS. The CCITT recommendation Q.921(I.441) is implemented in this package with the features of:

- Full implementation of the LAPD;
- Designed to maximize portability;
- Emphasis on standards compatibility;
- Voice and/or data call setup;
- File transfer on 16 and 64 kbps.
- Simultaneous voice and data call over two B channels with D channel signaling.
- The package provides interfaces across the SAPIs (Service Access Point Interfaces) between layer 2 and layer 3 using standard primitives which can be used to integrate the upper layer programs with the ISDN Express card and the LAPD package.

Since this product is only implemented up to LAPD, the high layer functions and B channel packet-switched protocols have to be implemented by the user.

The DGM&S ISDN "S" Interface Co-processor Adapter is another product based on the IBM PC [DALE87A][DALE87B][DALE88]. the adapter is an intelligent communication subsystem employed as the ISDN basic rate interface. Functionally, this ISDN adapter board contains the co-processor subsystem, PC processor interface, dual-port subsystem, ISDN interface, and an analog interface. The functional diagram of this ISDN board can be found in [DALE87A], [DALE88]. This adapter implements the hardware functions required to support the CCITT I-series S reference point interface. It is designed to run as a slave processor in the PC. This relieves the host CPU of much of the ISDN communication function. The board contains no ROM. ISDN I.451, X.25 layer 3 and LAPD/B software are downloaded from the main processor via the dual-port RAM to a one-megabyte dynamic RAM. With the ISP188/ISDN basic Access Product which is a general purpose product in the sense of both hardware and software [DALE88], users can design, develop and implement their applications or products. One of the applications developed over the DGM&S ISDN product is the Lachman ISDN-LAN bridge [LACH88] which provides a LAN interconnection through ISDN channel with the TCP/IP.

3.3 A Session-layer Interface: NetBIOS

3.3.1 The IBM version

Concepts

The IBM's Network Basic Input/Output System (NetBIOS) is the Application Program Interface (API) that lets the IBM PC programs explicitly access LAN facilities (refer to Figure 3.4). The NetBIOS first appeared in 1984 [IBM84A] with the IBM PC Network adapter card designed for IBM by Sytek Inc.[SYTE86]. The IBM PC Network was IBM's first LAN which has mentioned in the previous section of this chapter. It is possible for any program to use the network while making the usual call to the MS-DOS (Microsoft Disk Operating System) file system through NetBIOS commands. Loosely speaking, NetBIOS is a programming gateway to sets of services that allow computer applications and devices to communicate.

Typically, the NetBIOS is designed for the data exchange in the LAN environment. However, two applications within the same machine can also use NetBIOS for data commu-

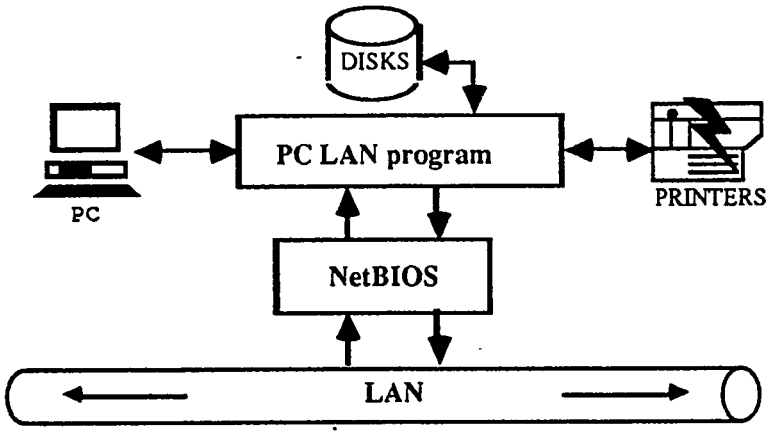


Figure 3.4: NetBIOS as the application interface

nication without a LAN. Thus the use of the NetBIOS is not restricted in LAN environments.

NetBIOS presents a fixed interface between the PC's operating system and the LAN. In concept, NetBIOS is similar to IBM's earlier BIOS (Basic Input/Output System), which provides an interface between the operating system and hardware components (discs, printers, etc.) of PC. In the same way that the BIOS freed the programmer from having to implement several different versions of a program to reflect the differences in the underlying hardware, so the NetBIOS frees the programmer from the same task for each different network.

Relation to OSI

Since the NetBIOS interface was designed to include many communications tasks that are not dependent on the operating system, the NetBIOS interface has its coverage extended to the Session layer. In terms of the OSI Reference Model, the NetBIOS is an interface between the Session layer and the Presentation layer as shown in Figure 3.6, and is thus a peer-to-peer interface.

The Datagram Support commands let applications contact the Data Link layer directly. In theory, any protocols at the Transport layer and below could be run under NetBIOS. An extended version of NetBIOS developed by Bell-Northern Research and Northern Telecom is implemented on the X.25 network layer protocols and ISDN protocols. It allows LAN software to be applied in the ISDN environment.

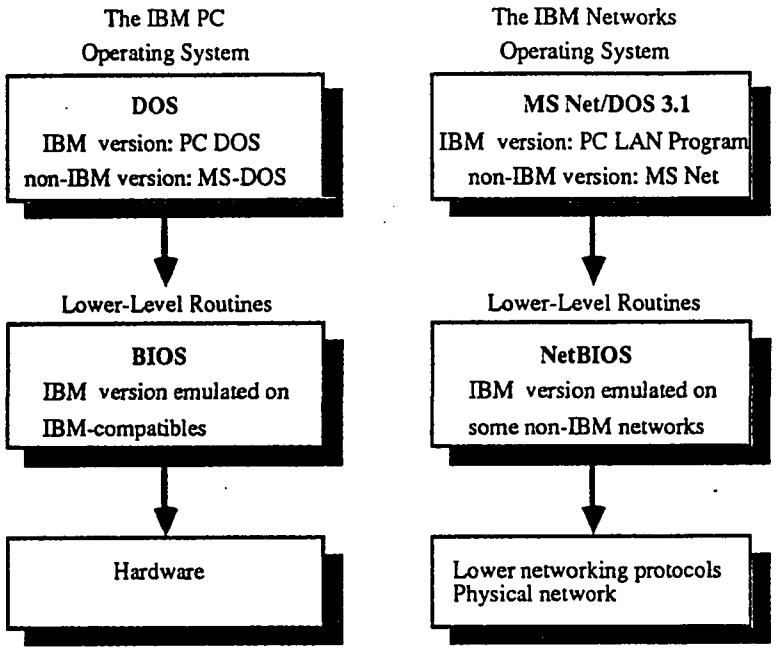


Figure 3.5: The relationship between NetBIOS and BIOS in the IBM family

Commands

NetBIOS contains only 19 commands which can be divided into four categories:

- *General-purpose commands* which are used to enable the adapter on the network, read status, or to control other outstanding commands.
- *Name Support commands* which allow actual user-chosen names to be used instead of network addresses for identifying calling and called nodes. Names have few restrictions.

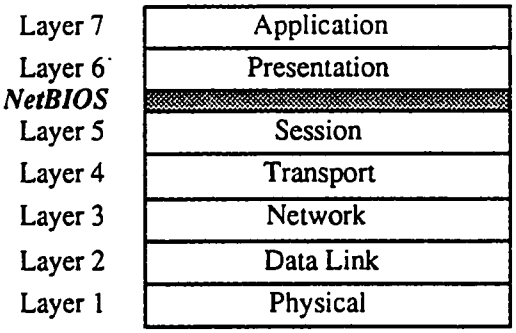


Figure 3.6: The NetBIOS interface relative to the OSI Model

they can be user names or the combination of names and numbers. A name can be unique, or the same name (called group name) can be added by many adapters on the network.

- *Session Support commands* which establish a logical connection (session) on the network, send and receive messages, terminate session and read session status. More than one command can be outstanding because the connection is in two-way simultaneous transmission mode.

- *Datagram Support commands* which support connectionless services allow the user to send a message to a name , group name, or to broadcast a message to all users. Messages are limited in size and never acknowledged by the receiver's adapter.

The description of the NetBIOS is found in [GLAS89], [SYTE86], and all commands of the NetBIOS are defined in [IBM84A].

All the commands and data operations to and from the NetBIOS are executed via a table called Network Control Block (NCB) (refer to Figure 3.7), which consists of 12 functional fields. The COMMAND field is used to pass a NetBIOS command from the application to the NetBIOS, and the RETURN CODE field is used to pass the executing result from NetBIOS back to the application. Two options of NetBIOS operation ('no wait' and 'wait') can be selected by the user. The no wait option allows multiple commands to be queued for execution within the adapter. The other fields in the NCB are largely concerned with session management. In the NetBIOS notation, a block of data passed across the NetBIOS interface is called a *Message*. This notation will be used in the later discussion. Figure 3.8 depicts an example of the exchange of the Message and the NCB across the NetBIOS interface between the Sytek LAN adapter and the host computer. In this particular NetBIOS implementation,DMA channels are used for NCB and Message transfer between the adapter memory and the host memory.

NetBIOS commands are independent of hardware and software. The same commands that worked with the original IBM PC network work with no change on Ethernet, ARCnet, Token Ring, StarLAN, and even simple serial-port LANs. A lot of application software can be run over the NetBIOS, such as Novell's NetWare, 3Com's 3+, Network OS, Banyan's VINES, Vianet, or any other IBM PC LAN software [GLAS89]. This combination of hardware and software independence makes NetBIOS an excellent vehicle for portable network software. The IBM has published the details of the NetBIOS interface [IBM84A] so that other manufacturers could produce NetBIOS emulations as well.

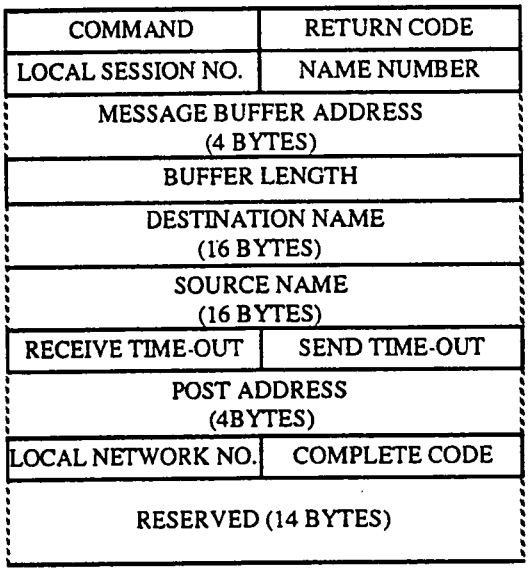


Figure 3.7: Format of NetBIOS Network Control Block (NCB)

3.3.2 An ISDN Version

The ISDN version of NetBIOS (referred to as PCTA's NetBIOS)[NORT88] is similar to IBM's NetBIOS standard. The PCTA's NetBIOS interface is also implemented by passing commands and associated parameters to the Session layer in a Session Control Block (SCB) [NORT88]. The SCB has the same function as, and a similar structure to the NCB defined in the standard NetBIOS (IBM NetBIOS).

Certain commands, however, are not implemented (e.g., datagram) and some commands are added (e.g., digital phone and circuit switching service). Table 3.1 shows the NetBIOS commands supported in LAN and in ISDN.

From Table 3.1, it can be found that the commands are almost the same, if the Session Support is used. However, some fields of the SCB or the use of these fields are different from those of the NCB:

1. Although the definition in the first 50 bytes of the SCB are exactly the same as those of the NCB, the use of the Receive Timeout and Send Timeout fields in SCB are dedicated and handled by the network rather than set by the application. When a session command is issued, these fields are ignored.

2. In NCB, the last 14 bytes do not contain any meaningful information. Since the

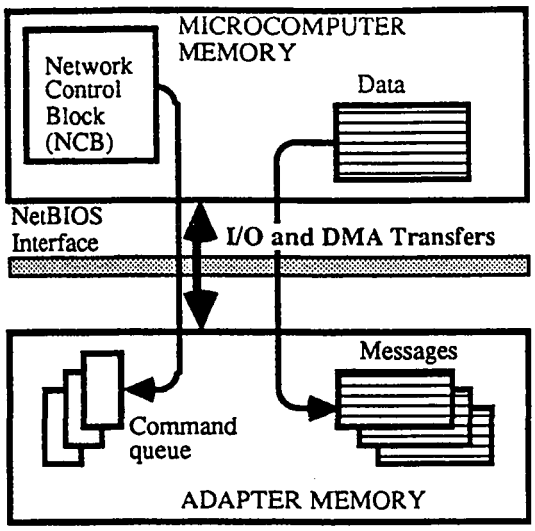


Figure 3.8: The 64-byte NCB allows the adapter to communicate with the host CPU

PCTA's NetBIOS is built on the X.25 protocol, 8 bytes which contains 1 byte of DATA CONTROL field, 4 bytes of BUFFER2 ADDRESS field, 2 bytes of BUFFER2 LENGTH and 1 byte of APPLICATION ID field are used for:

- requiring access to the X.25 packet-header Q bit
- passing up to 16 bytes of user data to X.25 in the CALL command
- setting application ID on a REGISTER command used for subsequent circuit-switching commands.

The detail descriptions of the PCTA's NetBIOS can be found in [NORT88]. Parts of the NetBIOS commands are not supported by the ISDN PCTA (refer to Table 3.1). Those commands are not implemented at the ISDN side of a gateway. If those commands are isssued at the ISDN side, they will be simply ignored by the ISDN PCTA.

3.4 The Application-layer Gateway

3.4.1 Motivation

According to the hardware and software of the subnetworks discussed above, we can summarize that the characteristics of the two LAN adapters are close. Each can replace the other.

	Commands of ISDN PCTA NetBIOS	Commands of IBM LAN NetBIOS	Code
General-purpose			
	CANCEL	CANCEL	35H
	no support	RESET	32H
	no support	ADAPTER STATUS	33H
	no support	UNLINK	70H
Name Support			
	ADD_NAME	ADD NAME	30H, B0H
	DEL_NAME	DELETE NAME	31H, B1H
	no support	ADD GROUP NAME	36H, B6H
Session Support			
	CALL	CALL	10H, 90H
	LISTEN	LISTEN	11H, 91H
	HANG_UP	HANG UP	12H, 92H
	SEND	SEND	14H, 94H
	RECEIVE	RECEIVE	15H, 95H
	RECEIVE_ANY	RECEIVE ANY	16H, 96H
	SEND_DOUBLE	CHAIN SEND	17H, 97H
	no support	SESSION STATUS	34H, B4H
Datagram Support			
	no support	*support*	
SAPI NetBIOS Extensions			
	support	*no support*	
Circuit Switching NetBIOS Extension			
	support	*no support*	

Table 3.1: NetBIOS: IBM version and ISDN PCTA version

The three ISDN adapters provide software interfaces on the different layers. Therefore, one of the following three approaches may be our internetworking solution.

First, the highest internetwork transmission rate can be achieved by a bridge in the Data Link layer where only the PDUs of the Physical and Data link layers have to be processed in the bridge. Therefore, the internetworking unit shown in Figure 3.1 may be implemented as a bridge (refer to Figure 3.9 (a)). In this case, all the LAN and ISDN adapters mentioned in the previous section of this chapter can be used. For the bridge solution, if a LAN communicates with an ISDN, the LAN stations and the ISDN stations should support the same protocols above the Data Link layer. Otherwise, the information exchange between a LAN station and an ISDN station cannot be achieved. But if ISDN only passes data streams from one LAN to another LAN rather than to the ISDN station, the upper-layer protocols in the ISDN can be different from LAN protocols.

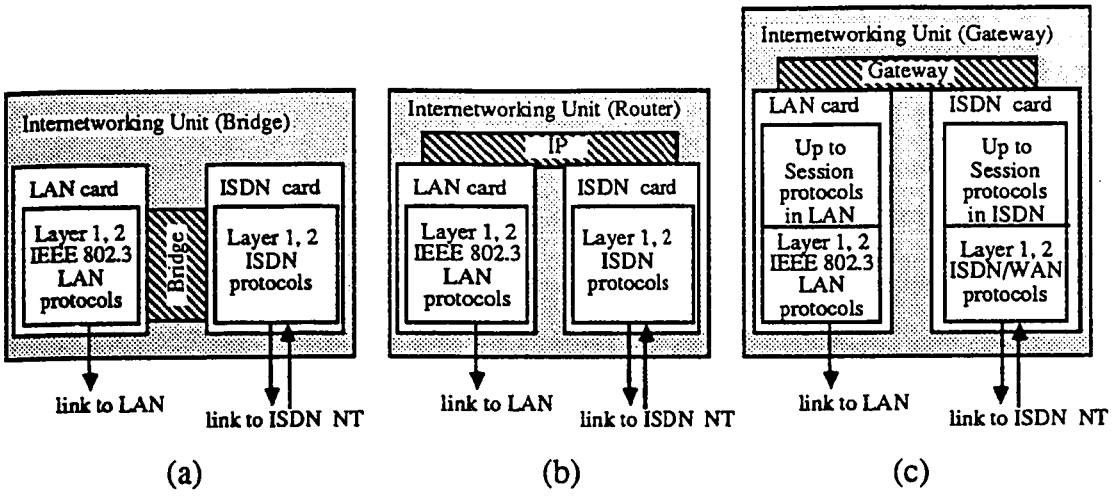


Figure 3.9: LANs interconnection through ISDN with a (a) bridge, (b) router, (c) gateway

Second, an IP router could be the internetworking solution (refer to Figure 3.9 (b)). In this approach, not only the IP protocol has to be added to end-stations and the routers, but also, a TCP/IP NetBIOS implementation has to be used instead of the original NetBIOS. This implementation requires NetBIOS extensions for resolution of internetwork routing in global environments.

Finally, an application-layer gateway is considered to implement the internetworking on the platforms of the Northern Telecom ISDN PCTA and Sytek (and Ethernet) adapters because they all support a NetBIOS interface in the Session layer (refer to Figure 3.9 (c)). This gateway is chosen for the following reasons:

1. *Applications are built on the NetBIOS.* The application software, such as the multimedia communication system [GEOR89] is built on the NetBIOS in the LAN environment.

2. *The Sytek Network adapter and the ISDN PCTA support similar interfaces between the Application layer and the Session layer.* Suppose that two heterogeneous networks with layered structures need to be interconnected. If a mappable interface at a layer can be found, the internetworking may be implemented using the cascading technique. If it cannot be found, a new layer has to be added in one or both networks (refer to Section 2.2).

   Since the subnetworks support the NetBIOS interface which has only 19 commands[9], the internetworking over this interface will be simple. Although the ISDN version of

[9]In the IBM's NetBIOS

NetBIOS is not exactly the same as IBM's, a set of identical functions between the two versions can be identified. Based on these identical functions, the data file transfer can be realized between the LANs via the ISDN or between one of the LANs and the ISDN. The only requirement of this approach is that LAN stations, ISDN stations and the gateway should support NetBIOS. This requirement is similar to that of the internetworking with TCP/IP.

3. *The application-layer gateway will not be the most serious bottleneck because the transmission rate on the ISDN is lower.* The Sytek LAN runs at a speed of 2 Mbps and the Ethernet at 10 Mbps. If two LANs are interconnected directly, the processing time in the internetworking unit should be as small as possible. In this case, the gateway is not suitable for high internetwork traffic because the gateway has to process multi-layer PDUs (protocol data units) internally. In some applications, this direct interconnection is still used if the internetwork traffic is not too heavy, for example the IBM used a NetBIOS-based gateway to interconnect the PC Network with the Token Ring Network [IBM86].

   Another possible LAN interconnection is via a WAN, such as an ISDN. Remote LANs are connected by a lower rate (64 kbps) long-distance channel. Since the transmission rate on the ISDN B-channel is lower (64 kbps), the data transmission time spent on the B channel will be longer than the data processing time spent in the gateway even if the gateway is on the Application layer. The bottleneck of the internetwork transmission will be the ISDN B-channel. Therefore, the gateway implemented using NetBIOS will be acceptable.

4. *The gateway does not need to change or modify the existing implementation of the LAN stations and the ISDN stations.* It is true that NetBIOS is the hardware/software independent package and can be implemented on any network. But the implementation is dependent upon the protocols on which the NetBIOS is supported. If the bridge in Figure 3.9 (a) or the router in Figure 3.9 (b) were employed using a Sytek LAN card and an ISDN PCTA card, the existing network software (below the Session layer) of end-stations in the LANs or the ISDN has to be changed.

3.4.2 The Gateway Structure

Although the NetBIOS was originally invented for a single network, internetworking can be implemented by means of the NetBIOS interface. It will make the gateway simpler and easier to be designed and implemented with 19 NetBIOS commands. This approach means it is necessary neither to modify the network adapters hardware and software nor to change existing applications. For an end-user in a LAN to communicate with another end-user in a remote LAN across the public network is the same as to talk with anyone else in the same LAN. This application-layer gateway structure contains the following components and is depicted in Figure 3.10.

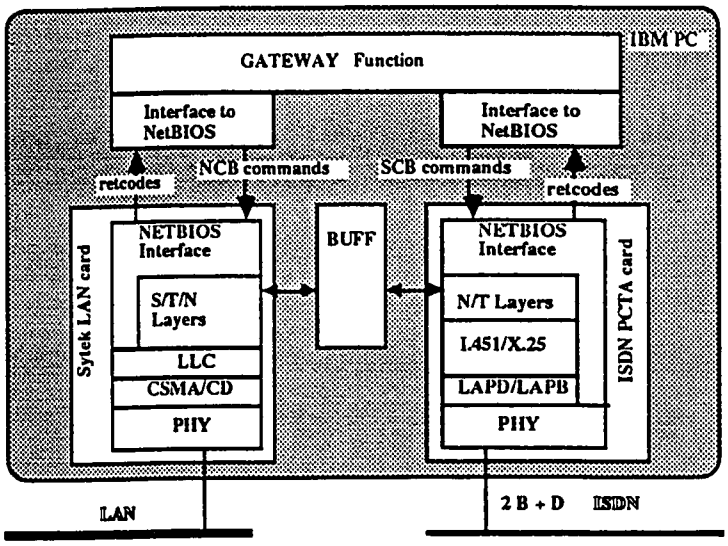


Figure 3.10: The gateway structure

- LAN side of the gateway: Sytek LAN adapter, or Ethernet LAN adapter.
- WAN side of the gateway: ISDN PCTA.
- The NetBIOS interface provided by both LAN and ISDN adapter cards.
- A gateway software, which will be designed and implemented.

The gateway shown in Figure 3.10 is dedicated as a relay between a LAN and an ISDN. The gateway program makes use of the NetBIOS interface to talk with both networks it

connects to. The NetBIOS can support up to 32 sessions simultaneously, so multiple sessions should be handled by the gateway. Since all adapters provide NetBIOS Session Support commands, the gateway is implemented with the conection-oriented service at both the LAN and the ISDN. Since the data file transfer must be reliable, this is a reasonable simplification.

The applications based on the NetBIOS Datagram Support interface may be transmitted through ISDN even if the datagram service is not supported by the ISDN NetBIOS. On the other hand, the ISDN circuit-switching on a B channel can be used as a link between two separate LANs. These issues will be discussed in the later chapters.

The features of the internetwork data transfer across the gateway will be:

- Data can be transferred on multiple sessions along the path of a source-destination pair. Data transfer based on the connection-oriented service is reliable. Therefore, no end-to-end session is required.
- Data can be transferred with mixed service modes, such as connectionless in the LAN and connection-oriented in the ISDN.
- Data transfer is in a stop-and-forward mode in the gateway, so there should be a data buffer for every session. The buffer also acts as a flow controller allowing data to be transmitted between a high speed LAN and a low speed ISDN.
- The internetwork routing is simply based on a name table in the gateway, so the difficult address translation from one network style to another can be avoided.
- Applications are transparent to the user.

3.5 Summary

In this chapter, an internetworking task for the existing applications is introduced. To find a proper solution for the internetworking, some characteristics of the existing Sytek and Ethernet LANs are analyzed. Then, some available ISDN adapters are evaluated and compared. Based on these adapters, three possible solutions are proposed. Among them, the application-layer gateway on the NetBIOS is chosen as the prototype in our trial according to the availability of hardware/software, the capability of the ISDN and the difficulty of the implementation. The features of the gateway are defined for a reliable data transfer between LANs through an ISDN link, or between a LAN and an ISDN. In the next chapter, gateway functions will be defined along with a state-machine specification.

Chapter 4

Formal Specification of the LAN-ISDN Gateway

A methodology which is used to realize the LAN/ISDN gateway generally contains three phases:*a specification phase*, *a design phase* and *an implementation phase.* In the previous chapter, the services and relative features of the gateway system have been defined. The gateway structure constructed includes the Sytek (or Ethernet) adapter and the ISDN PCTA card as the platforms and the NetBIOS as the interface. Therefore, the gateway system shown in Figure 3.10 can be simplified into three subsystems: a LAN-attached subsystem, an ISDN-attached subsystem and a gateway program (Figure 4.1). The hardware and software

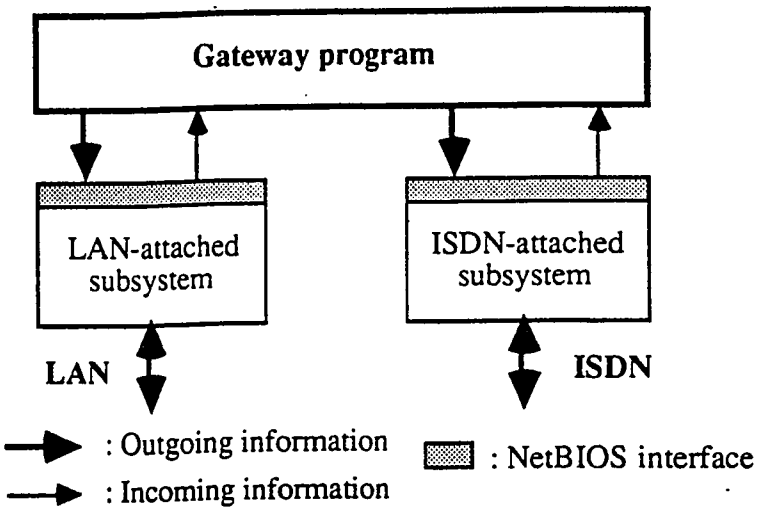


Figure 4.1: The gateway is divided into three subsystems.

interfaces of the two network-attached subsystems are the products ready at hand. The

peer-to-peer protocols of the subnetworks do not need to be specified since the protocols up to the Session layer have been implemented within the NetBIOS interface on the top of the Session layer [IBM84A]. We specify only the functions of the gateway program based on the services provided by the subnetworks attached. These two subsystems act as service providers of the gateway program. As for these subsystems, the only thing that has to be considered is how they can be configured when the gateway system is implemented.

The most interesting thing for us is to model the *gateway program* shown in Figure 4.1. The specification phase serves as a statement of the problem to be solved and the constraints limiting the designers' implementation options. In the design phase, a solution is developed that satisfies the specifications.

The gateway program interacts with two network-attached subsystems at the NetBIOS interface. In this chapter, the specification of the gateway program will be presented to define the gateway protocols which in turn define the functions of the gateway program. It will be modeled by finite state machine which is commonly used for protocol specification [ISO84] [HATL87]. It will be easier to design and implement the gateway system following a systematic description of the gateway protocol.

4.1 Interconnection Requirements

The gateway functions are dependent on services provided by subnetworks and the requirements of the application at an end station. In our session-layer gateway with the NetBIOS interface, two sets of internetwork communications may be classified according to the communication types within a LAN: connection-oriented communication between LAN stations, and connectionless communication between LAN stations.

4.1.1 End-to-End Communication over Sessions

The internetwork communication is based on the connection-oriented network services. Figure 4.2 depicts an end-to-end link which contains sessions along the path of a source-destination pair. When a user in one LAN wants to send data to a remote LAN via ISDN, first, connections within subnetworks should be established (steps 1, 2, 3 shown in Figure 4.2). Then, an end-to-end link acknowledgement may be applied dependent on the requirement of the end user (step 4 in Figure 4.2). The data can be transferred through several sessions (step 5 in Figure 4.2). Sessions are terminated after data transfer is completed. For

an end user, the internetwork communication should appear the same as the intranetwork communication. The user only feels that the transmission speed is slower for the case of an internetwork communication.

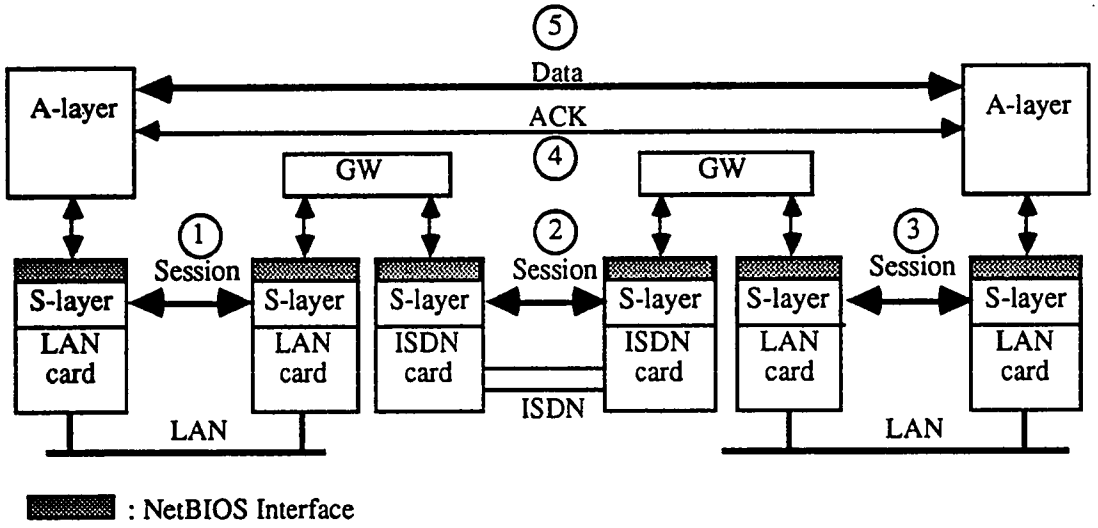


Figure 4.2: One logical link requires multiple sessions

4.1.2 End-to-End Communication with Datagrams

The connectionless service within a LAN can be extended to the inter-LAN environment. However, the ISDN NetBIOS does not support connectionless services as mentioned in the previous chapter. Therefore, the internetwork communication has to be based on mixed network services. Within LANs, the connectionless service is used. In the ISDN, the connection-oriented service is used as shown in Figure 4.3. Assume that a LAN station sends a short block of data to a remote LAN using a datagram. First, the data are sent by using the connectionless services from the sender to the gateway which is attached to the source LAN (step 1 in Figure 4.3). In ISDN, a session between gateways should be established (step 2 in Figure 4.3). Then the information is sent through the session (step 3 in Figure 4.3) After the data are sent to the gateway which is attached to the destination LAN, the gateway resends the data by using the datagram (step 4 in Figure 4.3).

This mixed configuration provides a solution for the inter-LAN datagram service. An end user in one LAN can send a datagram to other LANs through a connection-oriented link. However, the internetwork transmission delay is obviously large. Another problem may be

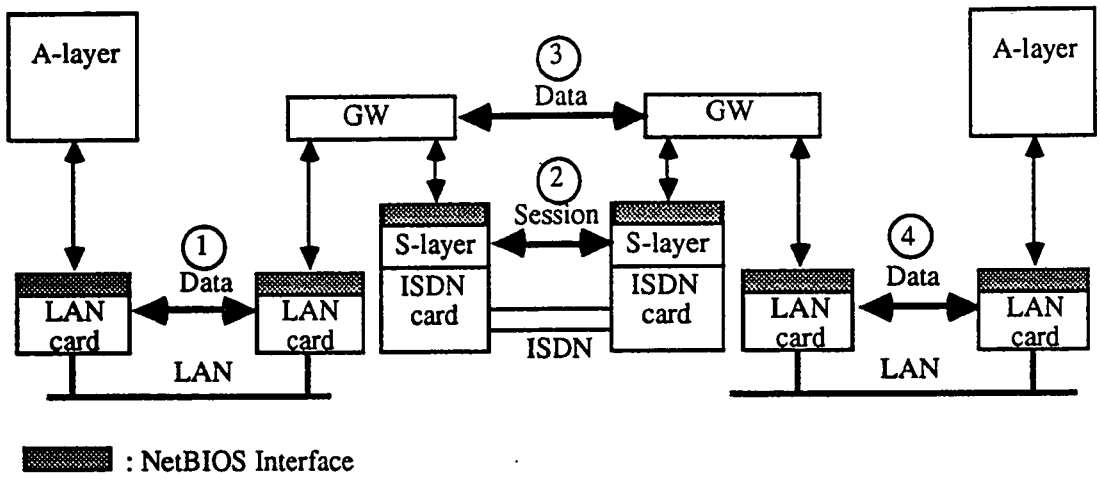


Figure 4.3: Datagram relayed by a session in ISDN

the time for setting up an ISDN session. If one active session is allowed to support several datagram messages, it is hard to determine the time-out value of the active session since the delay of datagram messages varies with the change of LAN traffic. However, it is too inefficient for a gateway to open and to close a session every time one datagram message is to be transmitted. The details of this set of data communication in the internetwork environment will not be considered in the thesis. Our attention will be focused on the connection-oriented internetwork data transfer.

4.2 General Aspects

According to the OSI Reference Model, the service is specified by a set of primitives available to a user or other entity to access the service at the service access point (SAP) (refer to Figure 1.2 (b) in Chapter 1). In our case, the service provided by the network-attached subsystems is specified by a set of NetBIOS commands and their return codes, which act as primitives at the interface between the Session layer and the Application layer. To transmit data on the subnetworks shown in Figure 4.2, the gateway has to be responsible for the establishment of two sessions, which are related to the same end-to-end communication link, at both sides of the gateway.

4.2.1 Assumptions

The gateway program is an application-layer program. However, the gateway program itself does not have to deal with the content of incoming and outgoing data. The function of the gateway program is to relay the data from one subnetwork to another. Therefore, the model of the gateway can be simplified. The gateway program interacts with its lower layers in two sides of the gateway as shown in Figure 4.1. Because the incoming and outgoing information are primitives rather than the protocol data units (PDUs), the term "Primitive" is used to denote incoming and outgoing information shown in Figure 4.1. The gateway function can also be depicted by a timing diagram (Figure 4.4).

In this figure, two vertical lines represent the NetBIOS interfaces. The X entity is a network-attached subsystem, and the Y entity is the peer network-attached subsystem. Such an entity can be a LAN-attached subsystem or an ISDN-attached subsystem. Between the lines, there is the gateway program referred to as the G entity which will be characterized later. The information exchange between the G entity and the entities are defined in terms of a set of incoming and outgoing primitives as shown in Figure 4.4.

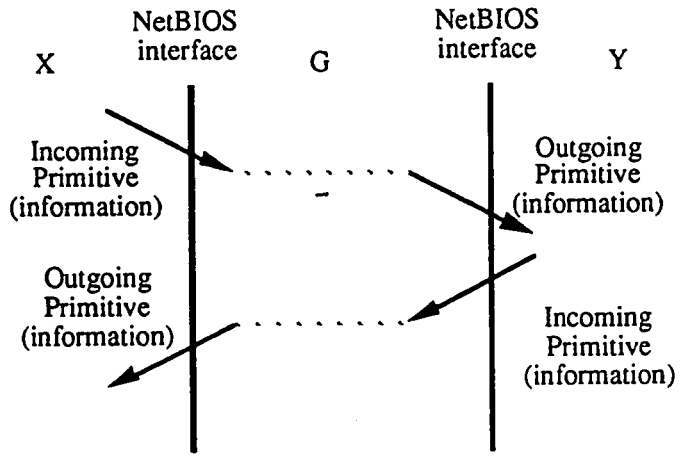


Figure 4.4: The operation at interfaces between three subsystems

We define notations *X(Y)_InPrim(M)* and *X(Y)_OutPrim(M)* to denote the incoming and outgoing primitives.

Outgoing primitive *X(Y)_OutPrim(M)* are issued by the G entity to the X entity or to the Y entity. where (M) is an optional parameter used for indicating any application data if there is any. Examples of outgoing primitives are X(Y)_CALL and X(Y)_LISTEN for session establishment on X or Y subnetwork, X(Y)_SEND and X(Y)_RECEIVE for data transfer and

X(Y)_HANGUP or X(Y)_RESET for session termination and initialization. These primitives may comprise some of the following parameters:

- *Called name*, name of the entity to which the session is to be established.
- *Calling name*, name of the entity issuing the information.
- *Send time-out value*, determining the time-out value during the data sent.
- *Receive time-out value*, determining the time-out value during the data received.
- *Session number*, numbering local active sessions.
- *Adapter number*, being issued for the G entity to recognize the X or the Y entity.
- *Buffer length*, indicating the size of an acknowledgement (ACK) or a Message (M). Acknowledgements and Messages are treated as the application-layer data by all entities in the gateway.
- *Buffer address*, indicating the location of an acknowledgement (ACK) or a Message (M) temporarily stored in the gateway.

Incoming primitive *X(Y)_InPrim(M)* are issued by the X (Y) entity to the G entity. The primitive indicates activities of both the subnetwork and the peer entity of an end station. Few parameters, for example a session sequence number will be contained in an incoming primitive.

Since the NetBIOS is used in our model, an incoming primitive can, however, be expressed as a return value of the outgoing primitive *X(Y)_OutPrim(M)_retcode* which has been issued to X (Y) entity. To simplify the procedure, the return codes (*_retcode*) of Outgoing primitive can be classified into five groups:

- *_ok*: This type of incoming primitive means that the previous outgoing primitive is accepted by the peer entity. and the operation on the X (or Y) entity (i.e., the operation on a subnetwork) is correct.
- *_closed*: This type of incoming primitives means that the session on the X (or Y) subnetwork is terminated.

- *_I*: The type I incoming primitives mean that a recoverable network error has happened at this moment. It can be simply recovered by reissuing the same command. *RECEIVE operation timed-out, message received incomplete, buffer length incorrect and no resource available* are classified as this type.

- *_II*: The type II incoming primitives mean that a network error can be recovered later but the established session(s) should be or has closed. *SEND operation timed-out, session ended abnormally, and de-registered a name which is active in a session* are classified as this type.

- *_III*: The type III incoming primitives mean that the established session(s) has to be closed and a network error has to be recovered by a system manager. For example, an incoming primitive *adapter malfunction* indicates that the adapter has detected an internal problem.

It should be mentioned that the G entity is not responsible for session operations, since it acts as an application-layer entity.

4.2.2 Definitions

The gateway program can be defined as a finite state machine whose outputs are determined both by its current and its past inputs [HATL87]. Commonly, a finite state machine is represented as a state transition diagram (Figure 4.5) and contains the following four components: *state*, *transition arcs*, *events* (i.e., incoming events) and *actions* (i.e., outgoing events).

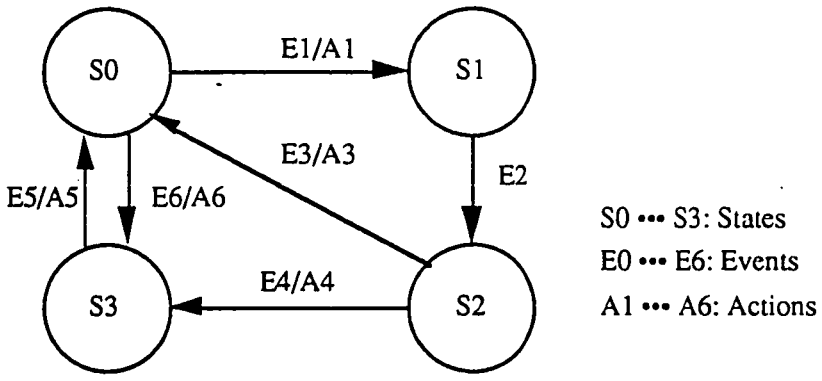


Figure 4.5: State transition diagram

To characterize the behavior of the G entity (i.e., the gateway program), those four components are defined as follows. In principle, the model we are developing here should be independent on any particular implementation. However, the NetBIOS has been chosen as interfaces between the G entity and the others which are located in lower layer than the G entity. Without loss of generality, all events and states are defined with notations close to the NetBIOS representation.

States:

Since the connection_oriented services have been chosen on both LAN and ISDN, the gateway has to establish virtual links (in our terminology, sessions) over both attached subnetworks for one end-to-end communication. Accordingly, states specified in our model should consider the conditions of both X and Y sides.

Transitions:

The transitions from one state to another are of the form of *Event/Action*. In our case, the event is the reception of an incoming primitive P, or otherwise defined. An action is the emission of an outgoing primitive E, or otherwise defined. As for the notation on the reception and emission primitives, *?P* means reception and *!P* means emission.

Events:

We define a notation *X(Y)_Event* to identify the events associated with the X (Y) entities. From the viewpoint of the G entity, an event in Figure 4.5 is the reception of an incoming primitive (*? X(Y)_InPrim(M)*) which is a return value of the previous outgoing primitive. We can further define *X(Y)_Event* as *? X(Y)_OutPrim(M)_retcode*. Two different primitives will be defined as two events, and two different return value of the same primitive will be defined as two events as well. The reception of one of the incoming primitives implies that the action(s) has taken from the peer entity of an end station. The event will cause the state machine to change its state and produce actions.

Actions:

A normal action *X(Y)_Action* is that the G entity issues outgoing primitives (i.e., ! X(Y) _OutPrim(M)) according to an event and the current state, for example issuing to X (or Y)

entity a CALL which attempts to establish a link with a listener on X (or Y) subnetwork. The listener may be another gateway or an end-user. However, the condition of this operation is that a link has been established in one side of the gateway. Without called by an end-user on one attached subnetwork, the gateway will never initiate a session on the other attached subnetwork.

Other actions associated with certain events and states can be classified as follows:

- *Action 1 (A_1)*: The transition to the same state is due to the reception of a type I incoming primitives which indicate errors. The action A_1 in terms of these events is to repeat the previous action. No parameter needs to be changed. A timer may be activated to avoid deadlock. If the timer expired, the action A_1 can be changed to the action A_2 or A_3 according to the current state.

- *Action 2 (A_2)*: The transition from the current state to state S_0 associated with the action A_2 is due to events which are associated with type II errors. If any active session of the end-to-end link is associated with one of these events, the action A_2 will instruct the X (Y) entity to close this active session. Then, a session-opening primitive for the same communication users will be issued again. One example of this type of errors is that the parameter may be changed accidentally.

- *Action 3 (A_3)*: The transition from the current state to the initial state is due to a event associated with type III errors. Since this type of events is corresponding to a serious problem on the network, the action A_3 may contain session closing, error reporting (e.g., display the error) and alarming. In this case, the recovery of the communication across the gateway is the responsibility of a gateway manager.

The implementation of an action is dependent on the current state. However, the differences will be not shown in the state machine which will be defined.

Although the subnetworks attached by the X entity and Y entity are different, those two entities are identical at the Session layer with respect to the connection-oriented services. Therefore, the X entity can be the LAN-attached subsystem or the ISDN-attached subsystem. So can the Y entity. However, if they are not identical, the prefixes X and Y have to be replaced by L (for LAN) or I (for ISDN).

4.3 Sessions on both LAN and ISDN

In this section, the finite state machine for the gateway program is defined.

Gateways help the establishment of an end-to-end internetwork link which contains several sessions. A gateway is concerned with two sessions of an end-to-end link. Since both versions of the NetBIOS (refer to Table 3.1) interface are identical at the Session layer, the session establishment on LAN and ISDN are identical as well. In this section, we will characterize the three-phase procedure for connection-oriented data transfer as we mentioned in Section 1.3 of Chapter 1.

For the convenience of analysis, we assume that a station in the subnetwork X is the caller and the sender, and a station in (or through) the subnetwork Y is the callee and the receiver.

4.3.1 Session Establishment Phase

The session establishment for every end-to-end link is independent of each other. Therefore, we only model the session establishment for a single end-to-end link.

Functions

1. The G entity waits for the session establishment with a caller in one subnetwork and establishes a consecutive session with a listener in the other subnetwork or in a subnetwork which is not attached to this gateway.
2. The application layer acknowledgement for the end-to-end link can be sent back to the caller through the gateway.

States

The states in the session establishment phase can be defined as follows;

- *INITIAL*: idle for any source-destination pair, and without pending listen commands for callers.
- *X pending, Y idle* (referred to as S_0): waiting for a call from a particular caller or from any caller (dependent on the parameter in the event ?X_LSN) at the X subnetwork, and the G entity does not issue any primitive to Y entity.

- *X open, Y opening*: session established in the X subnetwork, and waiting for session establishing from the listener in the Y subnetwork.
- *XY open*: the gateway has opened a session with the caller in the X subnetwork, the other session with the listener in the Y subnetwork.
- *XY open, waiting R(ACK)*: sessions established at both subnetworks without end-to-end confirmation, and waiting for receiving an end-to-end session established acknowledgement (ACK) from the receiver side (the Y side). The acknowledgement which is an application-layer message is different from the call acknowledgement which is a lower-layer signal. However, the application-layer ACK is not necessary since each session can provide reliable data transmission.
- *XY open, waiting S(ACK)*: waiting for the ACK being sent to X side on the condition of sessions opened at both subnetworks.

Events and Actions

The events and actions are related to the following outgoing primitives *X(Y)_OutPrim*:

- *X_LSN*: being established a session on the X subnetwork.
- *Y_CALL*: establishing a session on the Y subnetwork.
- *Y_REC(ACK)*: receiving the end-to-end acknowledgement from the listener at the Y side.
- *X_SEND(ACK)*: sending the end-to-end acknowledgement from Y to the caller at the X side.

The events are the receptions of the return values of the outgoing primitives, such as *?X_LSN_ok*, *?X_LSN_I*, *?X_LSN_II* and *?X_LSN_III* corresponding to the previous action *!X_LSN*. The definitions of other events are similar.

The actions in this phase can be defined as issuing the outgoing primitives: *!X_LSN*, *!Y_CALL*, *!Y_REC(ACK)* and *!X_SEND(ACK)*. Other actions A_1, A_2, and A_3 corresponding to the events *? X(Y)_OutPrim _I, _II, _III* have been defined in Section 4.2.2.

Automaton

Figure 4.6 depicts a state diagram for sessions connection of a single end-to-end link. In a normal condition, the state of the G entity should be in the S_0 state.

That means the gateway is waiting for an internetwork call from an end user. As soon as the G entity knows that a session from the caller to the gateway has been established correctly (i.e., the event *?X_LSN_ok* happens), the G entity tries to send a call to the other side of the gateway (i.e., the action !Y_CALL). The state will become *X open Y opening.* If the session has been established correctly, the gateway has opened two sessions which are related with an end-to-end link. In the state machine, the event *?Y_CALL_ok* turns the state into *XY open.* After that, two options can be selected.

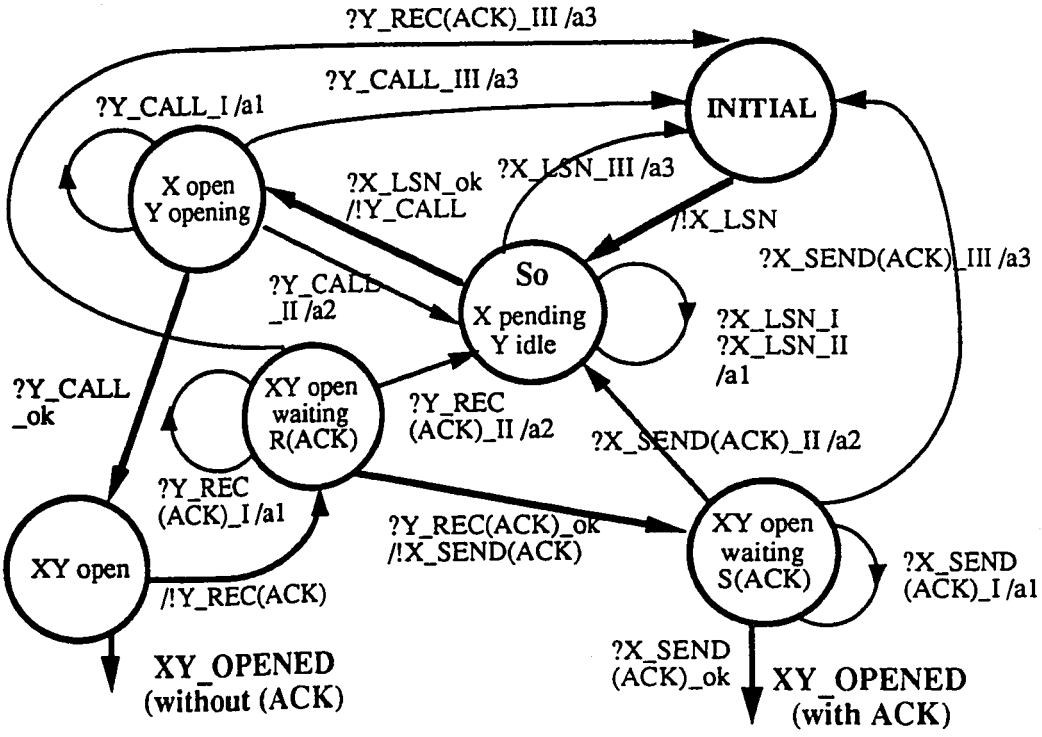


Figure 4.6: State diagram for the sessions establishment

Option I: The user of these sessions does not need the end-to-end link acknowledgement. The procedure can enter data transfer phase immediately (**XY_OPENED (without ACK)**).

Option II: The user of these sessions needs the end-to-end link acknowledgement. The procedure of the backward acknowledgement is required as shown in Figure 4.6. After

the action *!Y_REC(ACK)*, the state becomes *XY open, waiting R(ACK)* to waiting the *?Y_REC(ACK)_ok*. When this is done, the gateway should relay this information to the caller. The next state *XY open waiting S(ACK)* indicates that the G entity is waiting for the completion of sending the application-layer ACK. If an event is *?X_SEND_ok*. The end-to-end link is confirmed (**XY_OPENED (with ACK)**). The procedure goes in the data transfer phase which will be described later.

In the normal situation, the operation of the state machine is along the bold arcs as shown in the state diagram. If there is something wrong in a state, three alternative transitions with actions A_1, A_2, and A_3 are available as shown in Figure 4.6.

4.3.2 Data Transfer Phase

The data transfer is on the established sessions from one subnetwork to another in the form of the Message which has been defined in Section 3.3.1.

Functions

1. The G entity provides for an exchange of the Session layer data block (i.e., the Message) from one subnetwork to the other, and provides a buffer for Message storage.
2. The G entity does not require to know the content of the Message. It passes parameters obtained from an incoming primitive to an outgoing primitive, such as data size and buffer location.
3. On one end-to-end link, duplex data transfer is allowed. Since two ways transfers are independent and identical, it is enough to specify the data transfer in one direction in the gateway.

States

The data sender has been assumed to be from the X subnetwork.

- *receiving Mx*: waiting for a Message from the sender at the X subnetwork and storing in a Message buffer in the gateway.
- *sending Mx to Y*: waiting for the Message being sent to the receiver at the Y subnetwork.

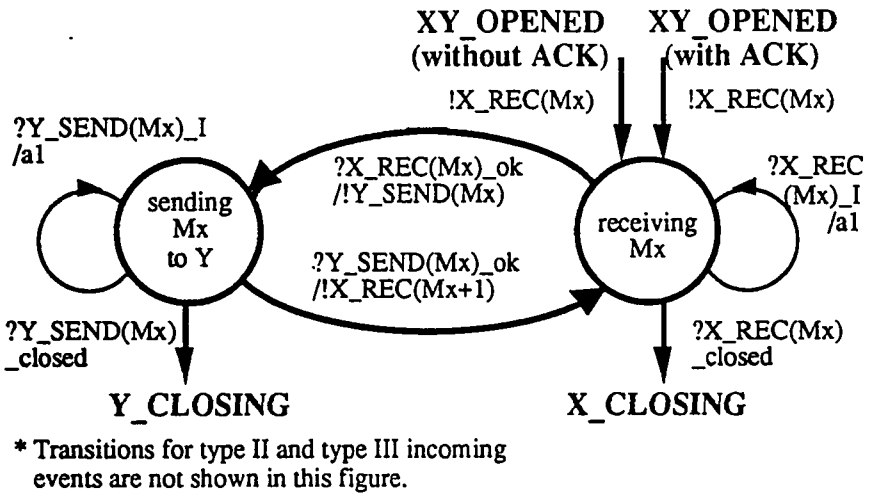


Figure 4.7: State diagram for the Message transfer

Events and Actions

The events and actions in the data transfer phase are related to the following outgoing primitives *X(Y)_OutPrim*:

- *X_REC(Mx)*: receiving a Message from the sender at the X subnetwork.
  X_REC(Mx+1): receiving the next Message from the sender at the X subnetwork.
- *Y_SEND(Mx)*: sending the Message from the gateway to the receiver at the Y subnetwork.

If duplex data transfer is required on the same logical link, there are two more incoming primitives in opposite directions:

- *Y_REC(My)*: receiving a Message from the sender at the Y subnetwork.
  Y_REC(My+1): receiving the next Message from the sender at the Y subnetwork.
- *X_SEND(My)*: sending a Message from the gateway to the receiver at the X subnetwork.

The events (for one-way transmission) *?X_REC(Mx)_ok, _closed, _I, _II* and *_III* are the receptions of the return values of X_REC(Mx). And the events *?Y_SEND(Mx)_ok,_closed, _I, _II* and *_III* are the reception of the return values of Y_SEND(Mx).

The actions can be defined as issuing the outgoing primitives: *!X_REC(Mx)*, *!X_REC(Mx+1)*, and *!Y_SEND(Mx)*. Other actions corresponding to the events *?X(Y)_OutPrim_I, _II, _III* have been defined in Section 4.2.2.

Automaton

Figure 4.7 depicts a state diagram for one-way Message transmission on sessions of a single end-to-end link. After sessions have been established on both sides of the gateway (denoted as **XY_OPENED (without ACK)** or **XY_OPENED (with ACK)** in Figure 4.6 and Figure 4.7), the G entity issues to the X entity an outgoing primitive to receive a Message from X side. The state becomes *receiving Mx*. Upon receiving an incoming primitive which indicates a data block (a Message) has been received correctly, the action is to issue the outgoing primitive to send the received data Mx to the Y side. The current state is changed to *sending Mx to Y*. If more than one Message (M) has to be transmitted, the state will be changed back and forth between these two states.

The transition to the session termination phase may start from the state *receiving Mx* as soon as the *X_REC(Mx)_closed* event occurs. It indicates that the termination is initiated by the sender (X side). The termination may also start from the state *sending Mx to Y* as soon as the *Y_SEND(Mx)_closed* occurs. It means that the termination is initiated by the receiver (Y side). The transitions based on the reception of the type II and type III incoming primitives exist but are not shown in the Figure 4.7 for simplicity.

4.3.3 Termination Phase

Functions

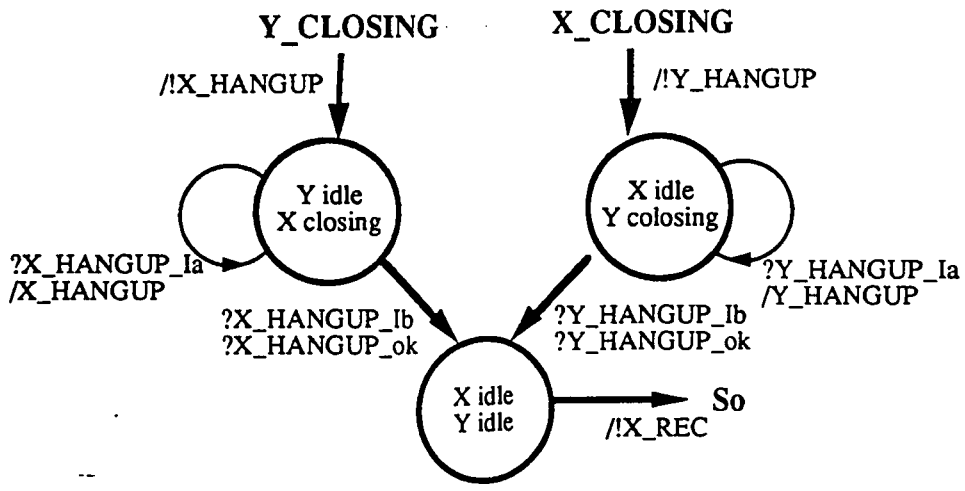
1. The G entity takes an action to instruct X or Y entity to terminate one session, when the other session has been closed by an end-user. The session termination is not initiated by the G entity in normal operation.

2. The G entity will, however, initiate a session termination, if it finds an error related to the session. This operation is a part of *action 2 (A_2)* and *action 3 (A_3)* which are described in Section 4.2.2.

States

The states in the session termination phase can be defined as follows:

- X idle, Y closing: sessions terminated at the X subnetwork, and terminating at the Y subnetwork.
- Y idle, X closing: sessions terminated at the X subnetwork, and terminating at the Y subnetwork.
- XY idle: sessions terminated at the X and Y subnetworks.



* Transitions for type II and type III incoming events are not shown in this figure.

Figure 4.8: State diagram for the sessions termination

Events and Actions

The events and actions in the termination phase are related to the following outgoing primitives *X(Y)_OutPrim*:

- *X(Y)_HANGUP*: closing an active session at the X (Y) network.

The events, *?X(Y)_HANGUP_ok, _Ia, _Ib, _II,* and *_III* are the receptions of outgoing primitive X(Y)_HANGUP.

The actions can be defined as issuing outgoing primitives: *!X(Y)_HANGUP*. Other actions corresponding to the events *?X(Y)_OutPrim_II, _III* have been defined in Section 4.2.2.

Automaton

Figure 4.8 depicts the behavior of the session closing. Since the event, which indicates the end-user has closed a session with the gateway, may be from X or Y entity, two parallel states have been defined. If an event indicates that the operation of the termination is correct, the data transfer on the end-to-end link is completed.

There are some differences on the error processing in this phase. The type I errors can be divided into two parts. One part (designated as part "a", for example, ?Y_HANGUP_Ia) does not guarantee the session is closed, it is necessary to reissue a HANGUP event. The other part (designated as part "b", for example, ?Y_HANGUP_Ib)) indicates the session which is to be closed has been closed. No action is required. The transitions in terms of the type II and and III errors for the states *X idle, Y closing* and *X idle, Y closing* are not shown in this figure.

For the next session establishment, the action *!X_REC* has to be taken, which forces the state into *So* again.

4.4 Summary

Two internetworking approaches may be used for our gateway implementation. One is based on the connection-oriented services of the subnetworks. The other is based on the combination of the connectionless service with the connection-oriented service in subnetworks. In this chapter, we have analyzed and specified the former approach. The state machines of session establishment, data transfer and session termination are proposed. The design and implementation of the internetworking system are based on the generic specification of this chapter. Although the specification is focused on a single end-to-end link, simultaneous session calls can be accepted and routed by the G entity (i.e., the gateway program). In the next chapter, the gateway program will be designed and a prototype of the gateway system will be implemented.

Chapter 5

Real-time Gateway Program

The specification phase serves as a statement of the problem to be solved and the constraints limiting the designers' implementation options. In the design phase, a solution is developed that satisfies the specifications. The implementation phase translates the design into reality. In this chapter, a gateway based on the connection-oriented services of the subnetworks is designed and implemented. Emphasis is put on the design of the gateway program and the implementation of the gateway system. As shown in Figure 4.1 in Chapter 4, the gateway program uses the NetBIOS to communicate with two attached networks.

5.1 Design of the Gateway

In the gateway design, some issues have to be dealt with, which include naming and routing, session handling, data transfer, multitasking, buffering and error handling.

5.1.1 Naming/Addressing

Three approaches for naming/addressing and routing have been developed in internetworking environments for quite a long time: *source routing* [PITT87B], *hierarchical naming/addressing* [COME88],[JANS87] and *name/address mapping* [JANS87]. Since the gateway is in the Application layer, names are used to represent users (or applications) on any network instead of addresses. Therefore, the terms *name* and *naming* will be mentioned rather than *address* and *addressing* in the later discussion.

The source routing approach does not need complex naming-table management. It does, however, require the direct and complete involvement of the end users in the internetworking scheme. One of the applications using this approach is the IEEE 802 LAN interconnection

by bridge which has been mentioned in Chapter 2. With source routing, one application would refer to another by means of a string containing the names (or addresses in the lower layers) of all the gateways on the route to that application.

The hierarchical naming approach has been discussed in Chapter 2. This is commonly suitable for large and multiple network environments. Although a name table is required for the internetwork routing, the size of the name table and the complexity of the routing is reduced because a gateway maintains the name table which contains only the network names rather than end-users' names.

By the name mapping approach, a name table is created to refer to stations on other LANs, that is, the gateway uses internal tables to map local names into its remote equivalent and to map remote names, which are in the remote LANs, into its local equivalent (refer to Figure 5.2). All the names are treated as flat names which have been discussed in Section 2.2. Without any modification, this approach can be applied for the gateway with the NetBIOS interface, since one advantage of using flat names is that a user name (or an application name) can directly be put into a NCB NAME field of the NetBIOS. Therefore, the flat name can be directly used for the session routing by gateways. The NetBIOS NCB contains a name string up to sixteen characters so that the name space is large enough to name users (or applications) without duplicate in the internetworking environment.

Among these three approaches, Name mapping will be chosen for our prototype gateway system for the following reasons:

- This approach would appear to be preferable as it involves only the gateways, leaving the end terminals unaffected;
- The size of the networks is small and topology of the networks is not complicated;
- User names can be directly filled in the name fields of NetBIOS. The gateway does not have to know the lower layer addresses or node numbers;
- No extra routing scheme is needed.

In this approach, a name table is needed in the gateway for the internetwork naming and routing. The structure of the name table is simple. Two sub-tables are contained:

- the Local Name table: user names of a local LAN listed in the ISDN side;
- the Remote Name table: user names of other LANs and user names of ISDN listed in the local LAN side;

Since all names used in both intra- and inter-network communications are full names, any name listed in the gateways should be a flat name. A name assigned to a LAN station or an ISDN station has to differ from any other name in an interconnected network (which includes LANs and ISDN). The NetBIOS can only find the name duplication in a single network but not in an interconnected network. It is the gateway manager's responsibility to check, if the name to be added is already used in other subnetworks within the interconnected network. Therefore, a pseudo-hierarchical name is recommended to avoid the name duplication. The name can be written in the hierarchical form. However, it is treated as a flat name by networks and gateways. The other consideration, when a name is assigned, is that the name space in the adapter card is limited to sixteen names at most. The number of names handled by the NetBIOS may be enough for an end-user but not enough for a gateway. When a LAN adapter card is used in a gateway, the names of this card represent the remote stations which will be involved in the internetwork communication rather than the gateway. In this case, up to sixteen names in a LAN can call (and be called by) remote stations[1].

Figure 5.1 (a) shows an interconnected network which contains three LANs which can communicate with one another. And Figure 5.1 (b) depicts name tables in three gateways.

In gateway A, all station names of LAN A are listed in the ISDN side (i.e. in the ISDN PCTA card), and all the station names of LAN B, LAN C and ISDN are listed in the LAN card (refer to Figure 5.1 (b)). Therefore, the stations of LAN B, LAN C and ISDN are the virtual stations of LAN A.

The fixed routing is applied by using the name tables in gateways to simplify the inter-network routing. It is, however, vulnerable to loading fluctuations and failures. Figure 5.2 shows an example of routing. The station WS A as a caller can set up an end-to-end link (or called an end-to-end session) with the station WS B. The end-to-end link which includes three sessions is established step by step from the station WS A to the station WS B.

The name mapping approach is suitable for the gateway with the NetBIOS interface because the name table will be limited by the NetBIOS in a small size. However, a large amount of work is required to initialize and maintain the necessary name mapping table in a gateway, if the table is very large.

[1] Assume one station may have more than one name.

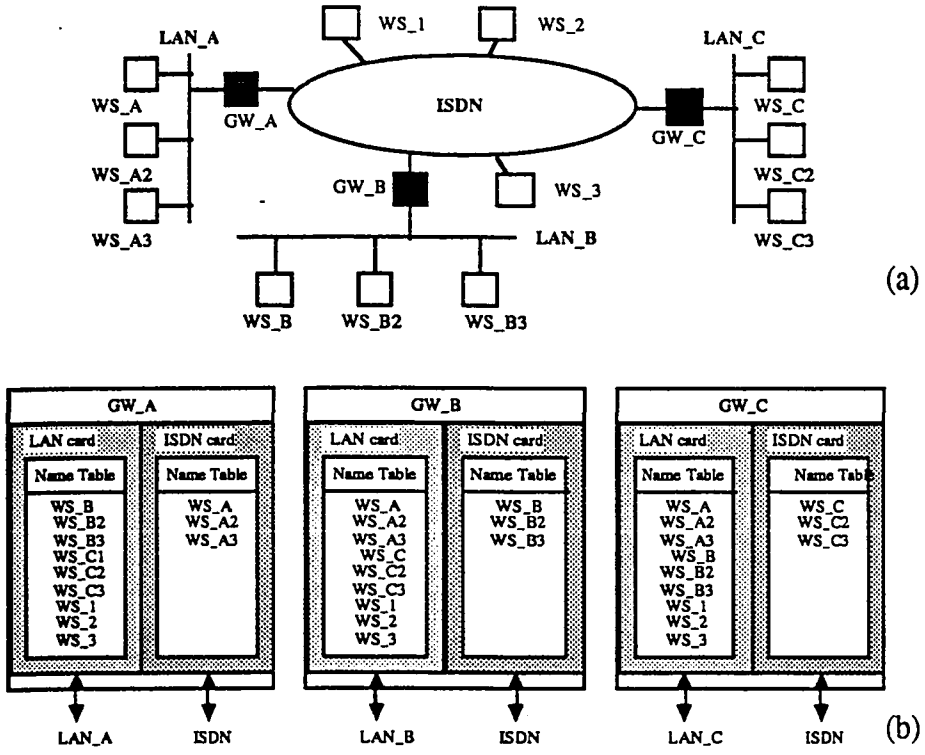


Figure 5.1: Name tables in the LAN/ISDN gateways

(a) An interconnected network (b) Name tables in three gateways

5.1.2 Data Transfer over Sessions

The main function of a gateway is to provide the reliable file and data transmission across multiple networks. Therefore, data transfer through an end-to-end logical link is required. Then, the end-to-end connection is a series of concatenated connections across gateways. A gateway should conform itself to this requirement, and have the capability of handling multiple connections.

Session Establishment and Termination

From the specification of the session handling in the gateway, a source-destination communication pair is on multiple sessions with an end-to-end session acknowledgement (shown in Figure 5.2) for more reliable communication. Either the end-user's application or an additional layer takes the responsibility for this acknowledgement. Therefore, the establishment

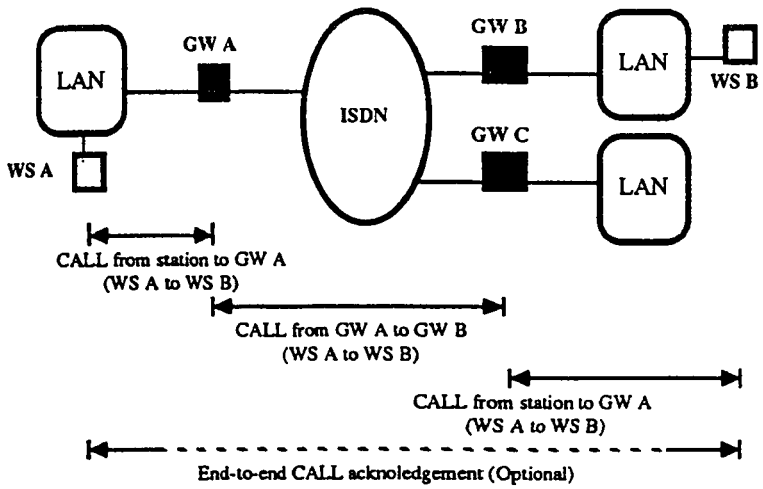


Figure 5.2: Routing of the name mapping approach

(and termination) of an end-to-end session (including multiple sessions) between source destination is realized with the help of gateways. Since the subnetworks used in our study support the NetBIOS, the end-to-end session acknowledgement is just a short data transfer from the listener back to the caller. It should be noted that the source must not be a caller only. The source may be a caller or a listener. If a station in LAN A in Figure 5.1 wants to communicate with another station in LAN B, the sessions within LAN A, ISDN and LAN B have to be established one after another. Figure 5.3 (a) and (b) show how the sessions are set up and closed between stations WS A and WS B using the NetBIOS session-support commands.

Initially, the station WS B has to issue a LISTEN command (LSN) to wait for a call from the station WS A. Since the station WS A sits in LAN A, the gateway GW B issues a LISTEN[2] command with the name WS B, and gateway A issues a LISTEN command with the name WS B as well. Before the station WS A issues a CALL command, all LISTEN commands are pending.

When the station WS A issues a CALL command which contains a called name (i.e. WS B), a session between the station WS A and the gateway GW A (as the virtual WS B) will be established, if both the station WS A and the gateway GW A receive a Good return code. Then, the gateway GW A (as the virtual station WS A) forwards the CALL to the ISDN to establish a session with the gateway GW B (as the virtual station WS B). Last, the gateway

[2]Commands with underline are issued in the ISDN side.

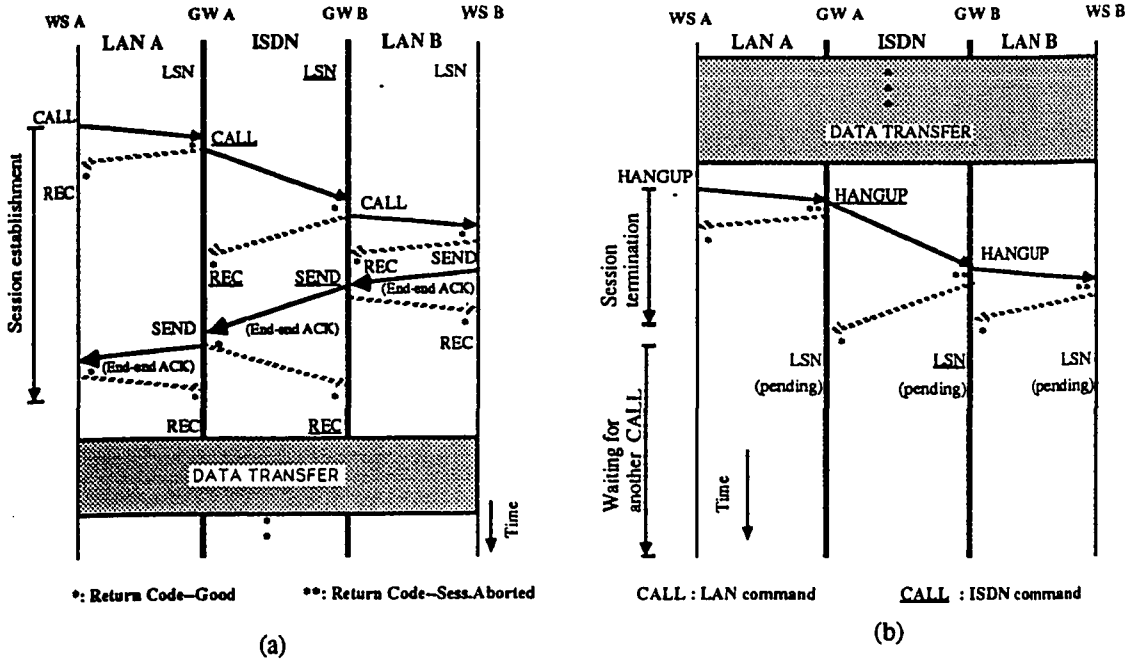


Figure 5.3: Procedure for session establishment and termination

(a) Session establishment (b) Session termination

GW B (as the virtual station WS A) calls the station WS B to establish a session in LAN B. After this time, data could be transmitted between stations WS A and WS B through these multiple sessions.

If the station WS A needs a response from the station WS B before file transfer, the gateways have to wait for an acknowledgement (end-end ACK) from the station WS B and send it in the reverse direction to the station WS A, i.e., after having completed the CALL operation, the caller and gateways issue a RECEIVE command (REC) to receive acknowledgement from the listener (refer to Figure 5.3 (a)). Actually the transmission of the session acknowledgement is the data transfer over the multiple sessions. In case the caller cannot receive the acknowledgement, the timer expiring will trigger an error recovering procedure to close all the sessions along a source-destination pair because there must be something wrong in the the end-to-end link.

Sessions can be terminated by either the station WS A or the station WS B after the data transfer. Figure 5.3 (b) depicts the session termination phase, and then the stations

and gateways are set to initial status.

The gateway program should have the following functions associated with sessions establishment and termination:

1. *Initializing a name table which maps all the remote stations.*

2. *Issuing LISTEN commands for all possible internetwork sessions.* If a name of an end station (preciously speaking, a name of an application) wants to have a session with a particular remote station, the LISTEN command should contain the name of the remote station. If the end station, such as a file server or a printer, will establish sessions with any other stations, these LISTEN commands are with a character "*" in the CALLNAME field of NCB instead of a particular station name.

3. *Relaying CALL commands to the other side of the gateway,* if a pending LISTEN is called from one side of the gateway and the returned code of this command indicates a session is established successfully between gateway and the caller.

4. *Keeping the session information for the data transfer,* such as:

   - Local session number, by which data are sent or received.
   - Time-out value, which decides how long a SEND or RECEIVE command will last.

5. *Terminating the session.* In normal situation, after data transfer, the gateway still has a RECEIVE command pending. When the end station issues a HANGUP command to close sessions. The gateway will be informed that a session is aborted by the return code of the last pending RECEIVE command. The gateway is responsible for closing the successive sessions of the same end-to-end link.

6. *Returning to the initial status.* After a session is closed, the gateway should issue another LISTEN command to wait next incoming call.

7. *Handling errors.* During the session open or close phase, the gateway program can report the error to the caller and directly return to the initial status in case an error happens.

Message Delivery

As mentioned in Chapter 3, a data block is called a Message in the NetBIOS terminology. A Message is transmitted in the stop-and-wait mode within sessions. This mode needs only one buffer per session in the gateway. Figure 5.4 (a) depicts the timing diagrams of the data transfer via two gateways GW A and GW B. The data are transmitted in the form of Messages.

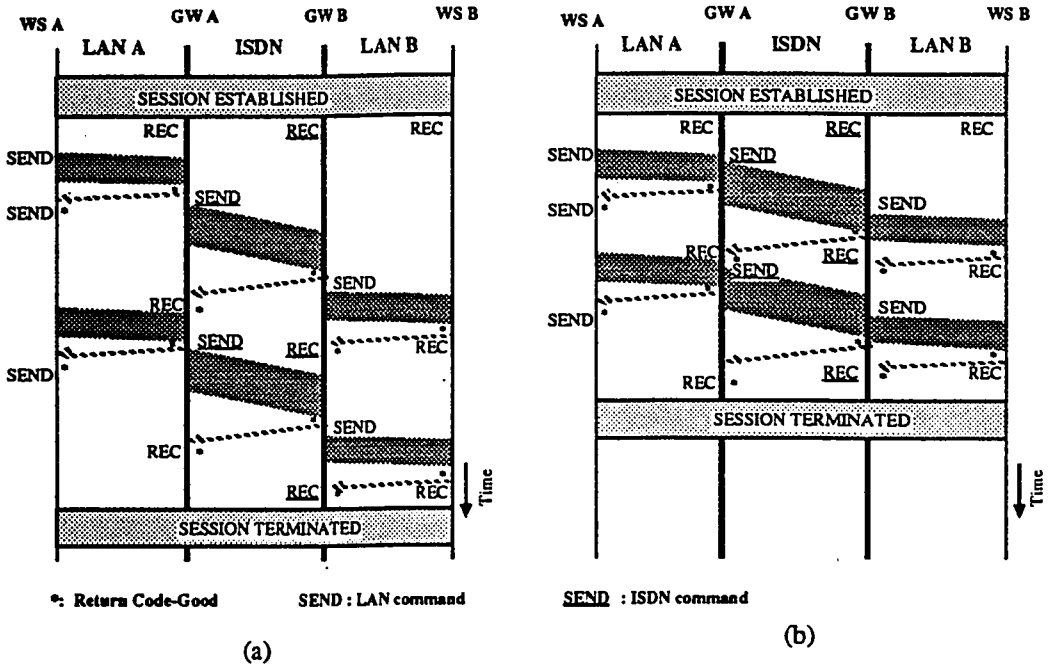


Figure 5.4: Procedure of data transfer

(a) Approach I (b) Approach II

Before transmitting data, the station WS B and the gateways GW A and GW B should issue a RECEIVE command (REC) to receive a Message. Since the sessions have been established, the RECEIVE commands use the session numbers which are in the SESSION NUMBER field of the NetBIOS NCB to recognize their proper sessions. It is necessary to use session numbers instead of names because several sessions may be associated with the same name. Since the local session number on the LAN side of the gateway is different from that on the ISDN side, it is the gateway's responsibility to keep track of the successive sessions. Then, the station WS A sends the first Message of a data file with the NetBIOS

command SEND. The pending RECEIVE at the gateway GW A will be performed. These SEND/RECEIVE commands are responsible for the reliable transmission of the first Message from the station WS A to the gateway GW A. After sending the Message, the station WS A issues another SEND command for the second Message transmission. This command, however, will be pending until the gateway GW A finishes sending the first Message to the gateway GW B and issues another RECEIVE command into the LAN side. The RECEIVE command issued by GW A will acknowledge the pending SEND command issued by WS A. From Figure 5.4 (a), it can be found that the second Message is received by the gateway GW A after the gateway GW A sends the previous Message completely. Followed this procedure, the data file is eventually transmitted to the destination.

If the gateway could send a Message after it received the first packet of the Message, the end-to-end transmission time would be reduced (refer to Figure 5.4 (b)). However, the data transfer on the session is on Message basis. It is impossible for the gateway program to know the moment when the first packet comes.

Since the transmission speed within the LANs is much higher than that in the ISDN, the transmission delay of Messages in LANs may be neglected. Therefore, the performances of these two methods are similar, but the implementation of the first method shown in Figure 5.4 (a) is simpler than that of the second one shown in Figure 5.4 (b).

From the above discussion, the gateway program should have the capabilities for data transfer as follows:

1. *Receiving a Message on a session.* Since all the lower layer functions are the responsibility of the LAN adapter and the ISDN adapter, a Message received by the gateway is error-free, if the NetBIOS informs the gateway program with a Good return code.

   For one way data transfer, if data are always sent from a caller or a listener, the gateway program only needs to issue a RECEIVE command on one side of the gateway. If the gateway does not know who is the sender, two RECEIVE commands are required on both sides of the gateway shown in Figure 5.5. One of the RECEIVE commands is always pending until the session is closed. Since the total number of pending commands is limited in the NetBIOS, it will reduce the number of active sessions supported by the gateway. Unfortunately, after all commands associated with a session have been issued, it is impossible to delete part of pending commands, since the NetBIOS does not support commands which can delete a data-transfer and still keep the related session active. However, the RECEIVE command in the reverse direction can be used

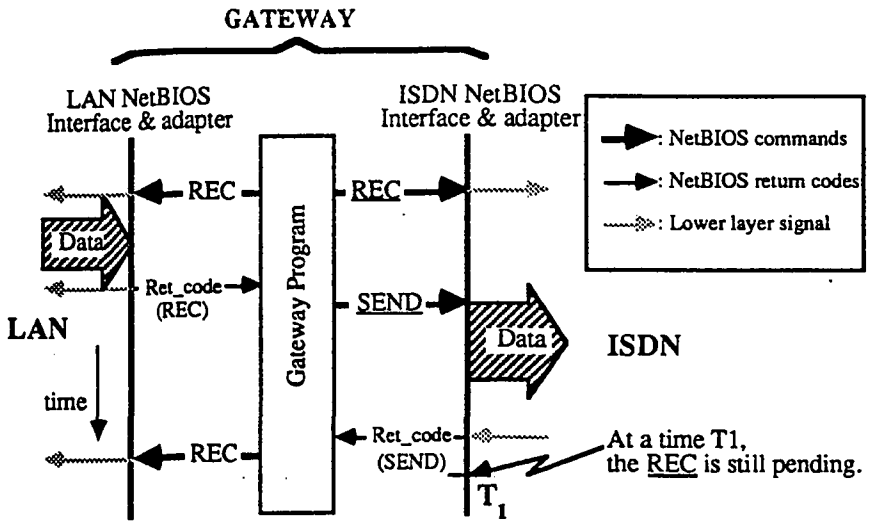


Figure 5.5: Two RECEIVE commands associated with the same sessions

as application-layer acknowledgement and remote-network status reporting. For the duplex data transfer, it is necessary to have two RECEIVE commands in a gateway after sessions have been established along a source-destination pair.

2. *Buffering the received Message.* A buffer is used for the Message storage in the gateway. With stop-and-wait operation, one buffer is enough for one direction of data transfer. The maximum buffer size (64K bytes) is limited by the NetBIOS.

3. *Relaying the Message and controlling the data flow.* The gateway program will not issue a SEND command to the other side of the gateway until it is informed by the NetBIOS that a whole Message has been received correctly. And the gateway program will not issue the RECEIVE command for the next coming Message, if the previous Message is not sent completely and correctly out of the gateway. This acts as flow control between the LAN and ISDN.

4. *Time-out consideration.* In gateways, the time-out values of the data transfer commands have to be large enough to wait for the peer commands response. In Figure 5.4 (a), a SEND command in the station WS A is issued as soon as the previous Message has been completely transmitted. The SEND command, however, has to wait quite a long time until the gateway GW A issues a RECEIVE command. In the ISDN PCTA, the time-out value is handled by the ISDN network [NORT88]. In the LAN adapter, the time-out value is specified with LISTEN and CALL commands.

5.1.3 Multitasking

The session operations including session establishment, session termination and data transfer are not limited in one end-to-end path. This will be convenient for some applications, such as data base accesses from several users simultaneously. Since the packet-switched service is used in both LAN and ISDN environments, a gateway can have parallel sessions established with stations or other gateways for the internetwork communications (refer to Figure 5.6).

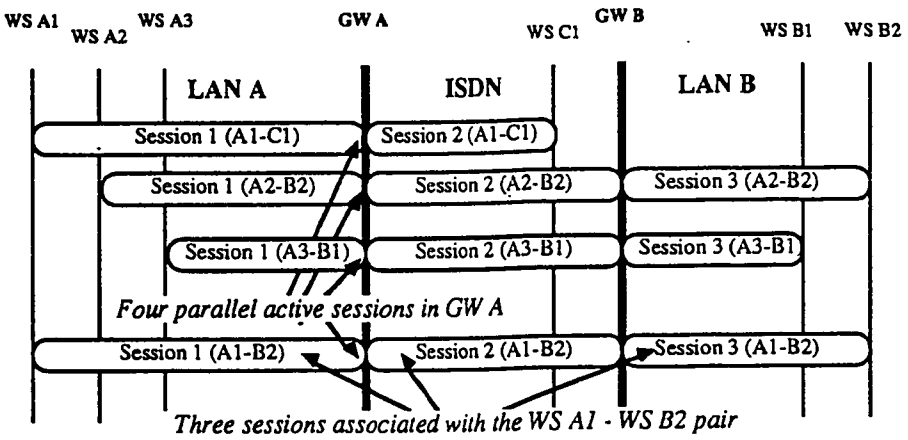


Figure 5.6: Parallel sessions in the gateway

The parallel-session operation in the gateway is referred to as *multitasking* so as to avoid confusion with the multiple sessions of a source-destination pair. The multitasking process in the gateway allows more than one internetwork data transfer at the same time. For example, if several internetwork source-destination pairs need to establish their end-to-end sessions, the gateway program should be capable of handling all the calls in real time.

The multitasking is realized by two parts in the gateway, one is the gateway program, the other is the two network adapters. The gateway program can issue multiple commands to one of the network adapters. The commands are then pending for execution within the adapter. As soon as one of the pending commands is executed or an error happens associated with the command, the gateway program will either relay the call or process the errors based on the return code from the NetBIOS. With the no-wait NetBIOS operation, the information (i.e., the command status) provided by the NetBIOS allows the gateway program to process multiple sessions. Normally, multi-processing or multi-computing jobs in realtime can be done by sharing one processor through *polling* or *interrupts*. These methods can be used for handling multiple tasks in the gateway program. Figure 5.7 gives an example of how several

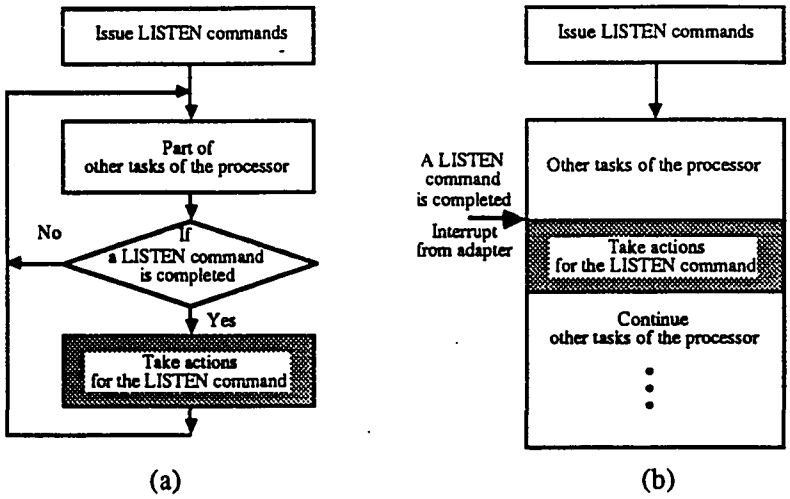


Figure 5.7: Parallel sessions handling

(a) Polling (b) Interrupt

sessions are established in one gateway. Assume that the gateway program has issued several LISTEN commands waiting for session establishment with different network stations.

1. *Applying polling.* In this method, the gateway program checks the return code in the COMMAND COMPLETE field of one pending LISTEN command to see if the status of the command has changed. If the status has changed, the gateway program takes proper action, such as establishing another session using the same name pair in the other network. When either the action for this LISTEN command is finished or it is still pending, the gateway program checks the next pending LISTEN commands in turn. A pending LISTEN command will be visited again after all other pending commands are polled. Figure 5.7 (a) depicts a simplified model of the polling method.

2. *Applying interrupts.* This method is an asynchronous operation. The gateway program dose not need to know whether the processing in the adapter is completed until the main processor of the gateway is interrupted by the adapter. It is the adapter instead of the gateway program that is responsible for checking the pending commands. Whenever a command is completed, the host processor stops executing the current program which is not in another interrupt routine, and executes the interrupt routine associated with the completed LISTEN command. Then the processor returns to the previous program.

Figure 5.7 (b) depicts a simplified model of the interrupt method.

In terms of efficiency, the interrupt method may be better than the polling one because it does not put the checking function in the gateway program. The response time will be shorter. If the host processor has to process a lot of tasks for the management of the gateway, or the number of pending commands is large, it is preferable to use the interrupt method. Because of the limitation of the ISDN channel capacity, it is not suggested that too many sessions be established for simultaneous data transfers. The dedicated gateway processor can handle multiple pending commands using the polling method as well.

5.1.4 Buffer Handling

One of the features of NetBIOS is that the application level does not know how the Message is fragmented into data packets in the lower layers. That is, every SEND or RECEIVE command handles one Message at a time. Therefore, the gateway program is only required to handle Messages instead of packets. Furthermore, the gateway acts as an intermediate node in the interconnected network so that it does not have to know the content of Messages. What the gateway needs to do is to pass the buffer pointer from the RECEIVE command on one side of the gateway to the SEND command on the other side of the gateway. A memory space should be reserved for the data buffering in the gateway. We have to determine how large a buffer should be and how many buffers are required.

- *The buffer size.* Since the gateway program sends and receives data in the form of a Message with the stop-and-wait mode, the buffer size should be equal to or larger than one Message size. The maximum size of a Message is limited to 64 kbytes by the NetBIOS.

- *The number of buffers.* In the gateway design, the number of buffers should be decided i.e., how the memory should be divided into buffers for active sessions. For every active session, the gateway must provide a buffer to temporarily store a Message. Otherwise, the data cannot be transmitted. *Static* or *dynamic* memory allocation can be used for buffer handling in the gateway.

The static memory allocation is that every possible session has its own Message buffer. The number of buffers equals the number of possible sessions. The total memory requirement is simply the sum of the Message size of buffers. If a link is fully duplex on an end-to-end

session, two buffers are required. The NetBIOS can support a maximum of 32 simultaneous sessions. Suppose all 32 sessions are used for one-way data transfer and the Message sizes of 32 sessions are all 64 kbytes. Totally, a 2048k-byte memory is required in the gateway. In most existing NetBIOS applications, the amount of data sent at one time is the configuration parameter, and is usually kept to less than 16 kbytes. Typical applications will only send 2 to 16 kbytes before waiting for an application-level acknowledgment. Thus, congestion, although possible in principle, is unlikely in practice [JANS87]. One way to reduce the memory size is to let end-users reduce the Message size. Another way is to share one buffer for those sessions which have little chance to be active at the same time.

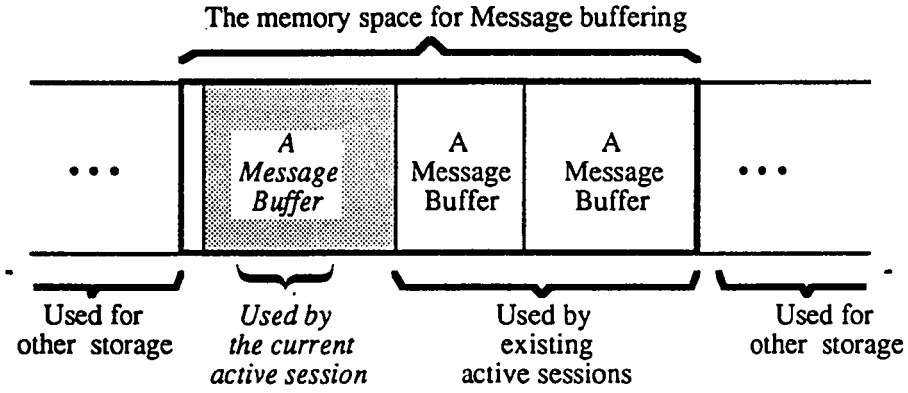


Figure 5.8: Dynamic memory allocation for buffering in the gateway

A more sophisticated way to handle buffers is the dynamic memory allocation. In the gateway, the dynamic buffer allocation means that a buffer is allocated to a session upon demand instead of preassigned to a particular session. When a session is not active, no buffer is reserved for it. Whenever an end-to-end session has been established, the gateway program checks the free memory space to find the starting address. If this part of the memory is larger than the buffer size of this active session, a buffer is obtained by the session (refer to Figure 5.8). The buffer size is dependent on the Message size of the current session. And the location of the buffer is dependent on previous active sessions and their Message sizes. In this way, the total memory space required is dependent on the probability of simultaneous active sessions. Dynamic memory allocation can be used in our implementation. For simplicity, a fixed-size buffer is assigned to each session rather than the dynamic buffer allocation. In this case, if a gateway supports up to 32 sessions, at least 32 buffers are needed. However, The size of a buffer may be small since the PC RAM which is used for Message buffering is not so large.

5.1.5 Error handling

Errors will happen because of incorrect operation, network malfunction, power-off of the called station, etc. If an error happens, the gateway program must be informed by the networks with a return code at the interface of the NetBIOS. The return code contains the network information.

If errors happen during installation, such as name-support command errors, the gateway program only displays an error on the screen to indicate what the error is. The same command with correct parameters is issued again.

If errors happen in the real-time environment, such as the session-support command errors, the gateway program not only reports an error in a log file or on the screen but also automatically recovers the normal operating condition. Based on the types of errors, the actions of the gateway program would be:

- Ending active sessions related to the same source-destination pair and back ack to the initial LISTEN pending status in case of *incompatible remote device*, *command canceled*, *command time out*, *session closed*, or *session ended abnormally* happened in any network in which the sessions are established.

- Trying the command at a later time in case of *too many commands pending*, *session open rejected*, *no resource available*, or *interface busy*.

- Exiting from the gateway program for reinstallation and reinitialization in case of *name not found*, *invalid adapter number*, *unusual network condition*, *adapter malfunction*, *session ended abnormally*, or *illegal buffer length*.

A case study for error recovery is described below. First, a logical end-to-end link is established between stations WS A and WS B through the gateway GW A and the gateway GW B with three sessions in LAN A, in ISDN and in LAN B shown in Figure 5.9. Suppose a Message is transmitted incorrectly, the station WS B cannot receive a Message completely from the gateway GW B. The station WS B gets this failure information with the return code of the RECEIVE command, and aborts the related session in LAN B by issuing a HANGUP command to LAN NetBIOS. There are two possible cases in the gateway GW B:

1. If the return code indicates that the sending Message has failed, the gateway GW B aborts the session in ISDN by issuing a HANGUP command to ISDN NetBIOS.

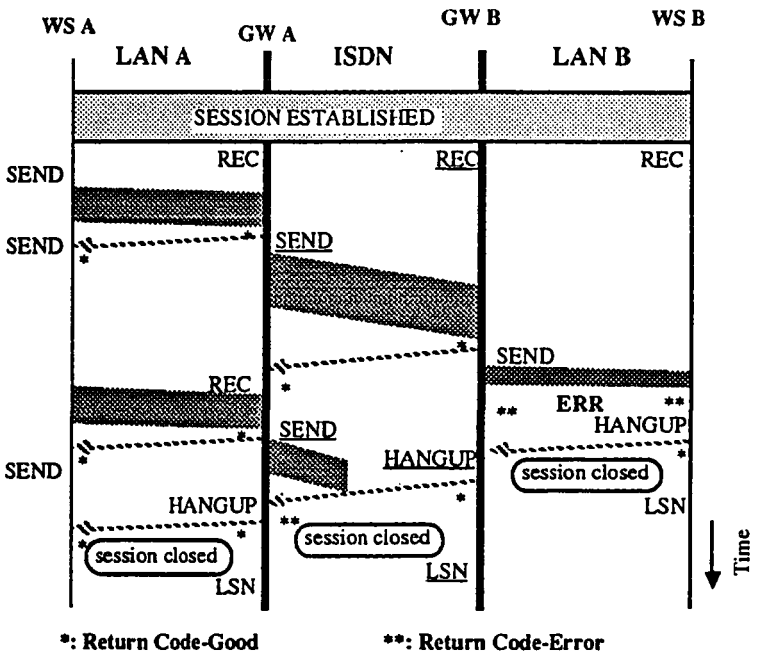


Figure 5.9: An error recovery procedure during the data transfer

2. If the gateway GW B does not get an error information from the return code of the current SEND command, the return code of the next SEND command will indicate the session on the LAN B has been aborted. Then, the gateway GW B issues a HANGUP to ISDN.

After the session in ISDN is closed, the gateway GW B automatically returns to the initial status, i.e., another LISTEN command will be issued to wait for the new call from the station WS A through the gateway GW A. Since the session in the ISDN has been closed, data cannot be sent from the gateway GW A to the gateway GW B. The gateway GW A is informed of the failure of the Message transmission by the return code of the SEND command and it aborts the session in LAN A with the command HANGUP. After the session in LAN A is closed, the gateway GW A returns to the initial status by issuing another LISTEN command. The HANGUP command issued by the gateway GW A informs the source station WS A to stop sending data. The application in the station WS A will take further actions, such as error report and reestablished sessions, etc.

If the pending RECEIVE command in the reverse direction has been used, the error in LAN B may be reported by gateways GW B and GW A back to the station WS A. The source station (WS A) can decide if the logical end-to-end link is terminated or not.

5.2 Implementation of the Gateway

5.2.1 System Requirements

This application-layer gateway structure contains the following components and has been depicted in Figure 3.10 in Chapter 3.

- Platform computer: IBM PC, PC-XT, PC-AT or PC compatible.
- Operating system: MS-DOS version 3.1, 3.2, or 3.3.
- PC memory (RAM) required for buffering and other resident software.
- LAN side of the gateway: Sytek LAN adapter, or Ethernet LAN adapter.
- WAN side of the gateway: ISDN PCTA.
- The NetBIOS interface provided by LAN and ISDN.
- A gateway program which has been designed.

5.2.2 Adapters Configuration and NetBIOS Installation

Before the gateway program is executed, the LAN and ISDN adapters have to be configured and the NetBIOS software has to be loaded into the LAN adapter and the ISDN adapter in the gateway.

Sytek

The hardware installation procedures of the Sytek LAN adapter and the RF translator unit are in [SYTE86] and [IBM84B]. The on-board jumper position settings is shown in Figure 5.10. When the power is on, the Sytek LAN adapter is automatically activated by a boot program. Since the NetBIOS is resident in the on-board ROM, no software installation is required.

Ethernet

The hardware installation of the Ethernet LAN adapter can be found in [WEST87]. The on-board jumper positions are set as shown in Figure 5.10. The NetBIOS is loaded to a shared

Jumpers	Jumper Position	Function
W1	In place	Automatic Network Loadin.
W2	In place	Option Jumper (reserved)
W3/W4	W3	IRQ 2
W6/W5, W7	W6	LANA_num = 0
W8	In place	On-board BIOS ROM
W9/W10	W10	32KB ROM for NetBIOS

Figure 5.10: Sytek adapter on-board jumper setup

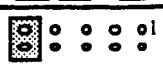
Jumpers	Jumper Position	Means
W1		I/O base Address: 280h
W2		Interrupt Request: IRQ3
W3	All in place	Network Type: thin Ethernet
W4	In place	Version:thin Ethernet
W5	In place	Segment Length

Figure 5.11: Ethernet adapter on-board jumper setup

memory with parameters by the program WPLUSNBE.EXE. The parameters should match on-board jumper positions. The principle of selecting the parameter and setting a jumper position is that no address conflicts with other devices in the PC (refer to a disk-based document INSTALL.DOC).

ISDN PCTA

No on-board jumper setup is required on the PCTA board. The hardware configuration procedures of the ISDN PCTA card are specified in [NORT87C].

The program ICONFIG.EXE provided with the PCTA card sets parameters before the NetBIOS is loaded [NORT87A]. In this adapter configuration program, there are 5 modules available in the main menu. For the back to back connection, parameters of gateways GW A and GW B are listed in Figure 5.12. Then the NetBIOS loading program can be run.

The PCTA NetBIOS software is selectable from four different packages in terms of the services required:

	Selection	GW A	GW B	Notes
A-Global Parameters:	TEI of the PC	2	2	Both should be the same.
	D-ch packet service	Yes	Yes	
	Provisioned B-ch	B1	B1	B1 or B2 selectable.
B-TA Hardware Driver:	I/O port	IRQ3	IRQ3	No conflict with PC devices
	DRQ Channel	1	1	
	TE or NT mode	TE	NT	Discuss in Chapter 6
	PC class	PC/XT	PC/XT	PC/AT or PC/XT.
C-X.25:	Send local address in Call Request packet?	Yes	Yes	Others are left in default values.
D-Session Layer:	Maximum Sessions	32	32	Range from 2 to 32
	LANA Number	1	1	Different from LAN adapter's
	Session Layter Name List Setup (see PCTA Adapter NameTable setup)			
E-Call manager	(not used)			

Figure 5.12: ISDN PCTA configuration in GW A and in GW B

- ISDN.EXE (basic NetBIOS capabilities including provisioned B channel packet-switching service capability)
- CISDN.EXE (basic NetBIOS capabilities including provisioned B channel packet-switching service capability and Call Manager application)
- QISDN.EXE (basic NetBIOS capabilities including B channel circuit-switching service capability)
- CQISDN.EXE (basic NetBIOS capabilities including B channel circuit-switching service capability and Call manager application)

The description of these programs is given in [NORT87A] and [NORT87B]. In our case, ISDN.EXE is chosen so that the packet-switching on B channel can be used. In the mean time, the packet-switching on the D channel can be used for data transmission as well.

5.2.3 Adapter Number Consideration

The adapter number (LANA_NUMBER field in the NCB of NetBIOS) is the parameter which allow the NetBIOS user (the gateway program or applications) to recognize adapter cards. The setup of the LANA number is different among three adapters we use:

- The LANA number of the ISDN PCTA has the default value 0. It is changeable to any value from 0 to 127 and is setup by a configuration program.
- The LANA Number of the Sytek LAN adapter can be selected either 0 or 1 by setting on-board jumpers.
- The LANA number of the Ethernet card is 0 and cannot be changed.

In an end station (either LAN station or ISDN station), it is better to set the LANA number to zero because of the limitation of some applications. In a gateway, the LAN adapter and the ISDN adapter should have different LANA numbers to be identified by the gateway program. Normally one is 0 the other is 1. If the Ethernet card is used in the gateway, the LANA number of the ISDN card has to be set to 1.

5.2.4 Prototype of the Gateway Program

Software Development

Based on the design of the gateway program, the flowchart of the program is shown in Figure 5.15. To simplify the programming, the following assumptions are made:

1. Flat names;
2. Session establishment without end-to-end acknowledgement;
3. All data transfer on active sessions;
4. Static buffer allocation;
5. Polling for multitasking.

The program is coded with the common high level language "C", and contains five functional source programs which are listed in appendix A. The implementation is divided into two modules, *Name table configuration*, and *real-time program running with error processing.*

Name Table Configuration

In the gateway program, the function of the name assignment following the name mapping approach is implemented. Two name tables are created. *The Adapter Name Table* contains

all the names which are associated with the possible internetwork connections. This table has been designed in Section 5.1.1.

In order to issue pending commands, another name table called *The Session Name Table* is created. It contains names which wait for the incoming calls from other subnetworks. In other words, this table has nothing to do with the adapter name assignment but determines the maximum number of active sessions. This table is only used in the gateway program.

Assume stations WS A2, WS B1 and WS B2 in an interconnected network shown in Figure 5.13 will have LISTEN commands pending. All LAN stations and an ISDN station can have internetwork calls. These two name tables in gateways are depicted in Figure 5.14 (a) and (b). The configurations of these two tables in a gateway are described below.

1. the Adapter Name Table configuration:

This part of the gateway program should have been implemented to handle both local and remote name tables in the gateway. Since the local name table which is located in the ISDN PCTA card is configurated by the ISDN configuration program ICONFIG.EXE, the gateway program is implemented only to add and delete remote names in the LAN card (Sytek or Ethernet). The program is implemented for a gateway manager to add and delete names in the Remote Name Table (in the LAN adapter) by means of a subroutine for the name setup. In our implementation, a name is added to or deleted from the LAN

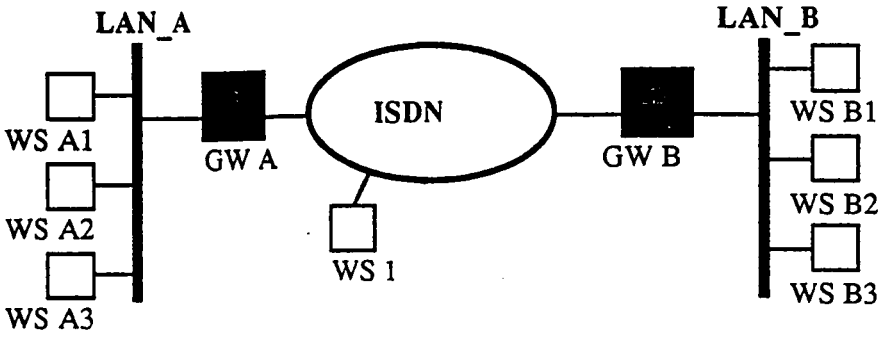


Figure 5.13: Two LANs are connected through an ISDN

adapter directed. A copy of the Remote Name Table is used to show the names on the PC screen. Whenever, the gateway program is reexecuted, the name table should be set up again. However, a disk-resident name file can be created to store the name table, if names in the table won't be changed frequently.

The ISDN PCTA system has its own name configuration program to add or delete names.

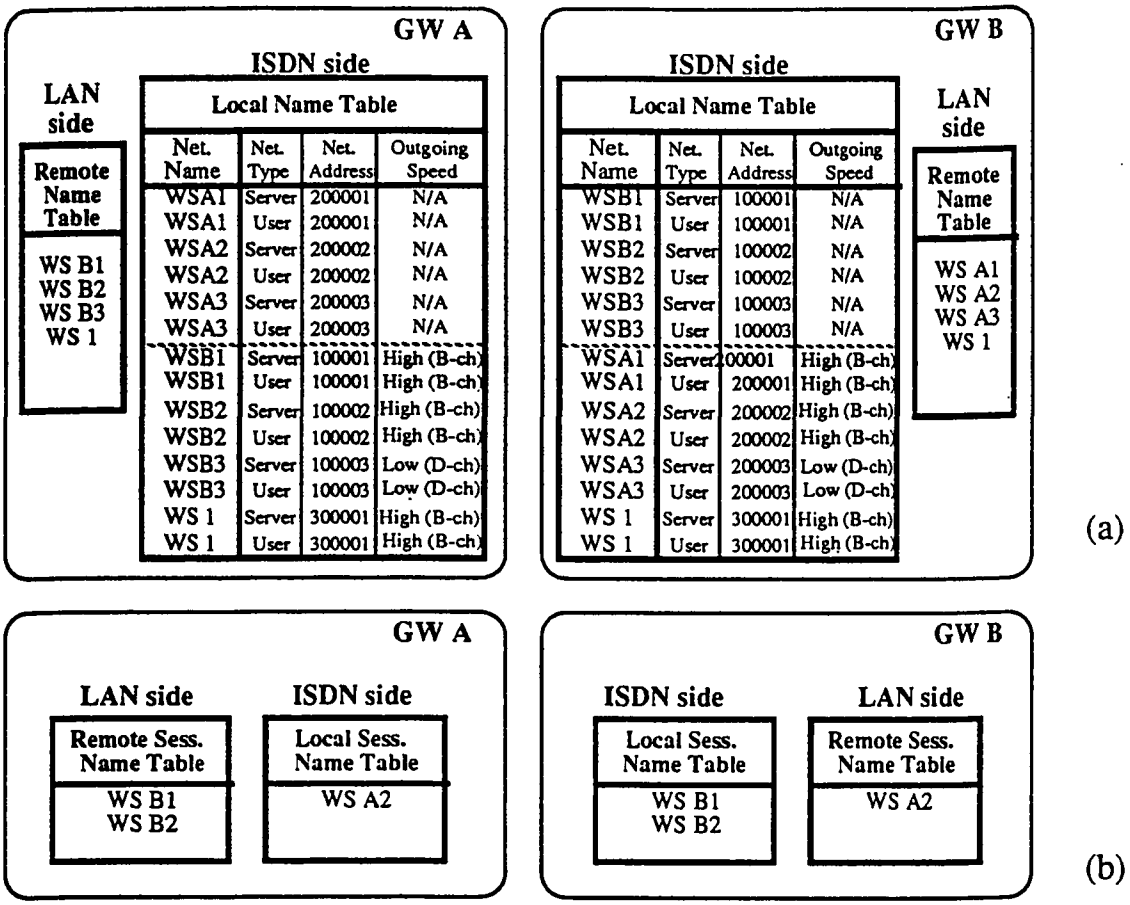


Figure 5.14: Configuration of the Adapter Name Tables and the Session Name tables

(a) The Adapter Name Tables (b) The Session Name Tables

In the notation of the configuration program, a sub-function called the Session Layer Name List Setup is used to list names in the Name Table of ISDN PCTA. In a gateway, a name list in ISDN PCTA is the Local Name Table as we designed above. In the ISDN PCTA, however, there are some special considerations in the name setup:

- The remote names and local names should be listed in the Local Name Table [NORT87A].
- The Name Table has been expanded to hold up to 256 different names instead of the 16 names in the LAN adapters.
- Every name has to be listed two times, one as a server and the other as a user in order to transmit data in both directions.

- A network address is assigned to a name.
- Outgoing speed is used for selecting the ISDN packet-switched channel (B channel or D channel). The principle is that if the name is in the local LAN, outgoing speed is not applicable. If a name is in the remote LAN, D or B channel can be selected.

If an ISDN station WS 1 wants to call stations of LAN A and LAN B, its name should be listed in the Adapter Name Table as shown in Figure 5.14 (a). Obviously, each station should have an Adapter Name Table which contains its name(s).

2. The Session Name Table configuration:

The Session Name Table is relevant to sessions. It is also divided into two tables, the Local Session Name Table and the Remote Session Name Table. In the ISDN side, the Local Session Name Table is filled with the name of local LAN's stations which will be called by stations in a remote LAN across the ISDN link or ISDN stations. In the local LAN side, the Remote Session Name Table is filled with the name of the remote LAN's stations or ISDN stations which will be called by any station in a local LAN across the ISDN link. Every name in these tables is associated with a LISTEN command. In the NetBIOs specification, the number of pending commands is limited to 32 and the number of the active sessions is limited to 32. If we issue 32 LISTEN commands in one side of the gateway, when all these commands are called, both sides of the gateway contain 32 sessions. In this case, the other side of the gateway cannot have a LISTEN command pending. In practice, the total number of the pending LISTEN commands in both sides of the gateway should be less than or equal to 32. The Session Name Table set up routine can check whether the total number of the names in both Remote and Local Session Name Tables is full.

Real-time Program running

After all the parameters have been set correctly, the gateway program can be executed until any key is hit. The function of this module is shown in Figure 5.15. First, the program finds the internetwork listeners from the Session Name table. After all LISTEN commands are issued, the pending commands are polled in the fixed order (shown in Figure 5.15 (a)). If the return code of a LISTEN command is checked, the program enters the session process routine as shown in Figure 5.15 (b). The status of the LISTEN command (from the return code) will be kept as a flag to route the program make the proper decision.When this completed LISTEN command is polled again, the program will poll the other commands within the

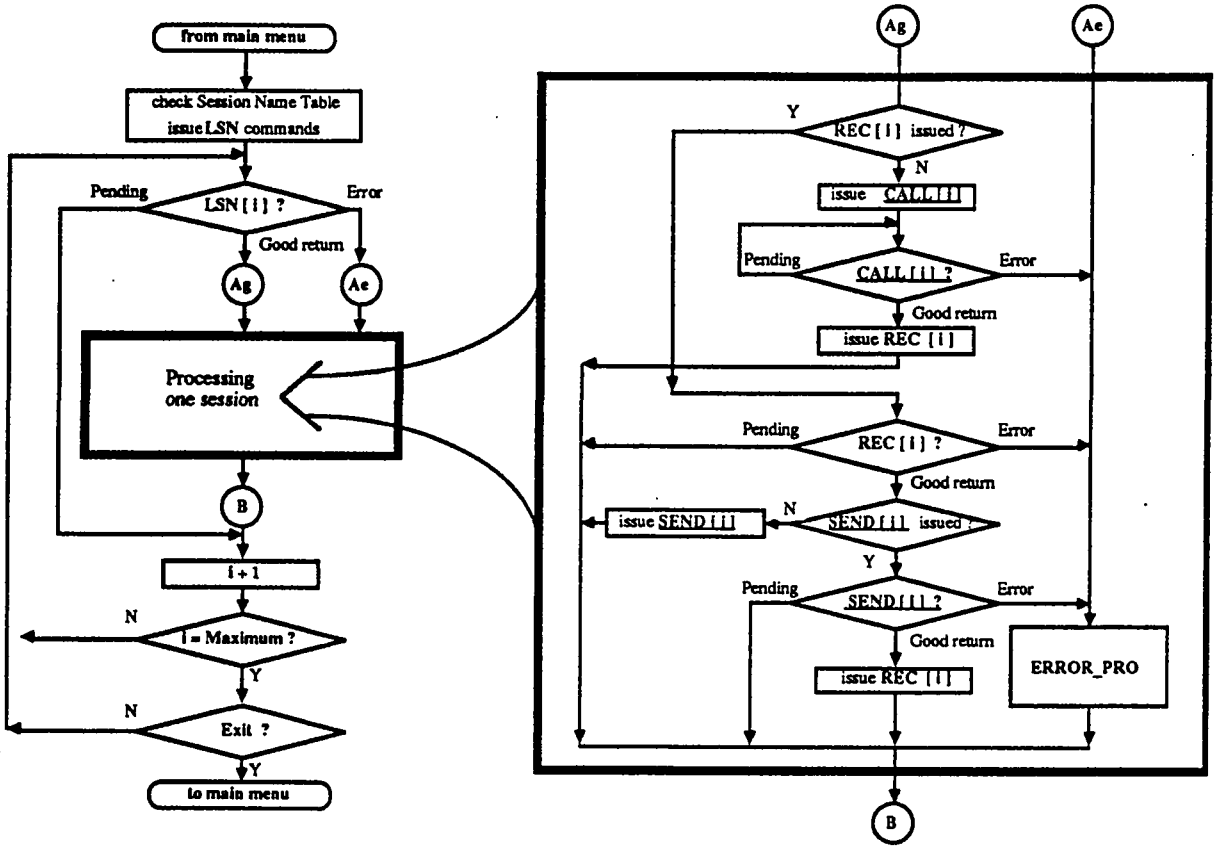


Figure 5.15: The gateway program applying the polling method

session process routine.

One drawback of the polling method is that the polling delay has to be added in the response time of commands because when a command is completed in this time, the gateway program will not know it until this command is polled. This is acceptable, if the pending commands are not too many.

There are over 30 different return codes indicating the errors of the NetBIOS commands. We have discussed some of the proper actions in the gateway program according to return codes. In our prototype program, a simple error processing procedure is implemented in the ERROR_PRO subroutine. Whenever the gateway finds or is informed of an error within a session, the gateway closes the other sessions which are related to the same end-to-end link, and issues a LISTEN command again. If this happens during the data transfer phase, the end user has to reestablish an end-to-end session and retransmit the data file. This operation does not affect other active end-to-end links supported by the gateway.

5.3 Summary

In this chapter, an application-layer gateway program has been designed for the internetwork data transmission. Since the data transmission is based on the connection-oriented network services, multiple sessions are required between the source station in one network and the destination station in another network. A session is only responsible for the reliable data transmission within one subnetwork. By applying the stop-and-wait protocol to session-layer data transfer, gateways help to relay data from one subnetwork to another. End-to-end acknowledgement for sessions establishment or for data transfer can be implemented, if end users need more reliable data transmission.

For a particular function, the implementation is not unique. More than one solution have been analyzed in the design. The choice is based on the real application requirements.

In the next chapter, a set of tests will be presented to test file transmission and to show the performance under different network conditions.

Chapter 6

Benchmarks

In the previous chapters, we have described the design and implementation of a LAN-ISDN gateway on the NetBIOS. All applications which interface with the NetBIOS can work within this internetwork environment. The gateway which is proposed in this thesis is on the Session layer. Therefore, it is natural that data transfer is slower through the gateway than through a lower layer relay. There are two ways to evaluate the performance of a gateway system. One is *simulation* which can be done by setting up a relative queuing model and giving parameters of the system. The discussion of the simulation is beyond the scope of this thesis. The other is *benchmarks* by which the performance of a real system can be measured. The term *benchmark* comes from the land surveying. In the computer field, a benchmark is a set of executable instructions which may be used to compare the relative performance of two or more computer systems. In our case, we measure the speeds of inter- and intra-network data transfers and compare the results with the channel capacity in different network conditions. Emphasis will be focused on the internetwork transmission speed of a file transfer in different cases.

6.1 Test Environment

6.1.1 Back-to-back Connection of Two Gateways

In an interconnected network as shown in Figure 6.1 (a), if all gateways are attached to the ISDN at the T reference point (refer to Figure 1.9 in Chapter 1), these gateways and ISDN stations act as the TEs (terminal equipments).

However, two gateways in our internetworking system are back-to-back connected with an ISDN link, since the tariffed ISDN is not placed in service (refer to Figure 6.1 (b)). The

ISDN link at the ISDN T reference point is asymmetric in accordance with the CCITT Recommendation [CCITT84]. The user side is the TE and the network side is the NT (network termination). Therefore, one gateway acts as a TE and the other gateway has to act as an NT. Two ISDN stations can be connected back-to-back by setting up one as the TE mode and the other as the NT mode as well. The selection of these two modes does not affect the operation of the gateway program.

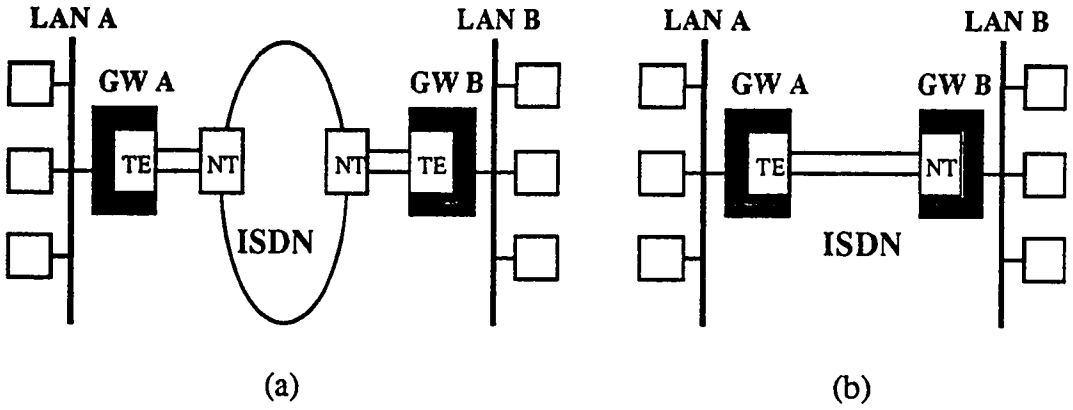


Figure 6.1: Interconnected network

(a) Normal gateway connection in an ISDN (b) Back-to-back gateway connection

The back-to-back connection of two gateways is configurated with an eight-conductor telephone line shown in Figure 6.2 [NORT87D]. 50 ohms resistors are for the line termination. The TE or NT mode is installed by a configuration program for the ISDN PCTA card (refer to Figure 5.12 in Chapter 5).

6.1.2 Stations and Gateways

In the internetworking system shown in Figure 6.1 (b), the devices used are as follows:

- A gateway, which was discussed in the previous chapters of this thesis.
- A LAN station, which is a PC and contains a Sytek (or Ethernet) LAN adaptor with the NetBIOS interface.
- An ISDN station, which is a PC and contains an ISDN PCTA card and an ISDN NetBIOS package. Since only two devices can be connected in ISDN (back-to-back

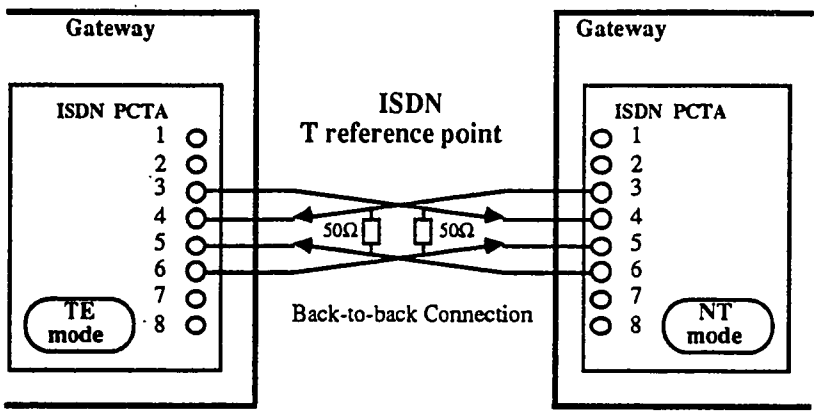


Figure 6.2: Back-to-back connection of two ISDN PCTA cards

connection), one gateway will be used as an ISDN station for the LAN-ISDN data transmission.

Since the LAN adapter and the ISDN adapter used in the gateway are the same as that of in a LAN station and in an ISDN station respectively, the hardware/software configurations for these adapters are the same as those of the LAN station and the ISDN station, which have been described in Section 5.2.

6.2 End-user Test Programs

The network behavior can be monitored by a network protocol analyzer. By means of this kind of analyzer, it is easy to find how fast a packet is transmitted to the network. In this case, we could measure the gateway efficiency when a real application is in operation. Since protocol analyzers or other custom testing equipments were not available, two test programs had to be developed for evaluating the performance of the end-to-end data transfer in the internetworking system. One is for disk-to-disk data transmission, the other is for memory-to-memory data transmission. In order to make the results comparable, all measurements are made with the same test program in different environments. The programs are implemented with the NetBIOS interface so that it can be used in the LAN or the ISDN environment directly.

6.2.1 Disk-to-disk Data Transmission

The disk-to-disk test program is used to transmit data from a hard disk (or a floppy disk) of a station to that of another station. The peer stations can be in one subnetwork or in different subnetworks. This program contains the following functions:

1. Selecting the LAN A Number: 0 or 1.
2. Adding and deleting a name in the Adapter name Table of a LAN station.
3. Opening and closing a session.
4. Transmitting data file
5. Resetting the LAN adapter card.
6. Opening and closing a data file in a hard disk.

The purpose of this test is to verify the relay function of the gateway program. A readable file and an executable file are under test, since these files can be easily checked. Especially, the executable file is very sensitive to errors. One error may result in the failure of the program execution. When the executable file is transmitted to a remote station, it can be executed on the remote station to see whether the transmission is correct or not. The disk-to-disk file transfer has been done (1) within a LAN, (2) from a LAN station to an ISDN station and (3) from a station in one LAN to a station in the other LAN through an ISDN B channel.

The measurement of the transmission time is not necessary under these circumstances because the hard disk access time, which is included in our measurement, is too large to get the transmission time correctly. If disk-to-disk data transfer on the ISDN network (Figure 6.7), a 40k-byte data transmission takes about 30 seconds. However, it only takes about 8 seconds to send the same data from one workstation to the other with memory-to-memory transmission.

6.2.2 Memory-to-memory Data Transmission

The memory-to-memory test program keeps all the functions of the previous program except the disk-access operation. Furthermore, in order to meet different test environments, the program allows for the selection of different Message sizes, for examples 1 k bytes, 2 k bytes,

4 k bytes, 8 k bytes, 16 k bytes, 32 k bytes or 64 k bytes. This alternative program is particularly used to test the end-to-end data transmission speed.

6.2.3 Time Measurement

Because of a lack of network testing tools, the behavior of the data on the network cannot be monitored. Therefore, the processing time of a gateway cannot be measured at the network interfaces shown in Figure 6.3 (a). Instead, we develop a test program by which the time of an end-to-end data transmission is measured at the NetBIOS interfaces shown in Figure 6.3 (b).

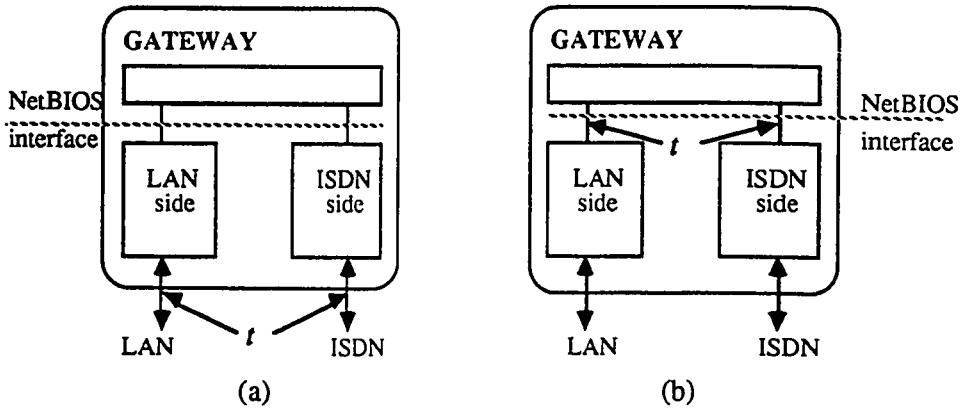


Figure 6.3: Two different ways to measure the transmission time

(a) Test points at network interfaces (b) Test points at NetBIOS interfaces

The test program acts as a NetBIOS user. As we have presented, before testing, the network adapters have to be set up, and an end-to-end session has to be established. Then, a data file can be transmitted for the measurement. When a sender sends the first Message of a file to the NetBIOS, a timer is started, and when the receiver gets the return code of the last RECEIVE command, the timer is stopped. The transmission time, then, is $T = t_{stop} - t_{start}$. The actual end-to-end transmission rate can be calculated by the equation: $R = \frac{Total\ data\ (in\ bit)}{T}$ bps. We find that, although a gateway is operated in the stop-and-wait mode, Message transfers across gateways are not exactly in the stop-and-wait mode, if an application-layer acknowledgement from the receiving station to the sending station is not used for every Message. After a Message has been sent from the sending station, the next Message is sent only after the nearest gateway sends the previous Message. It is

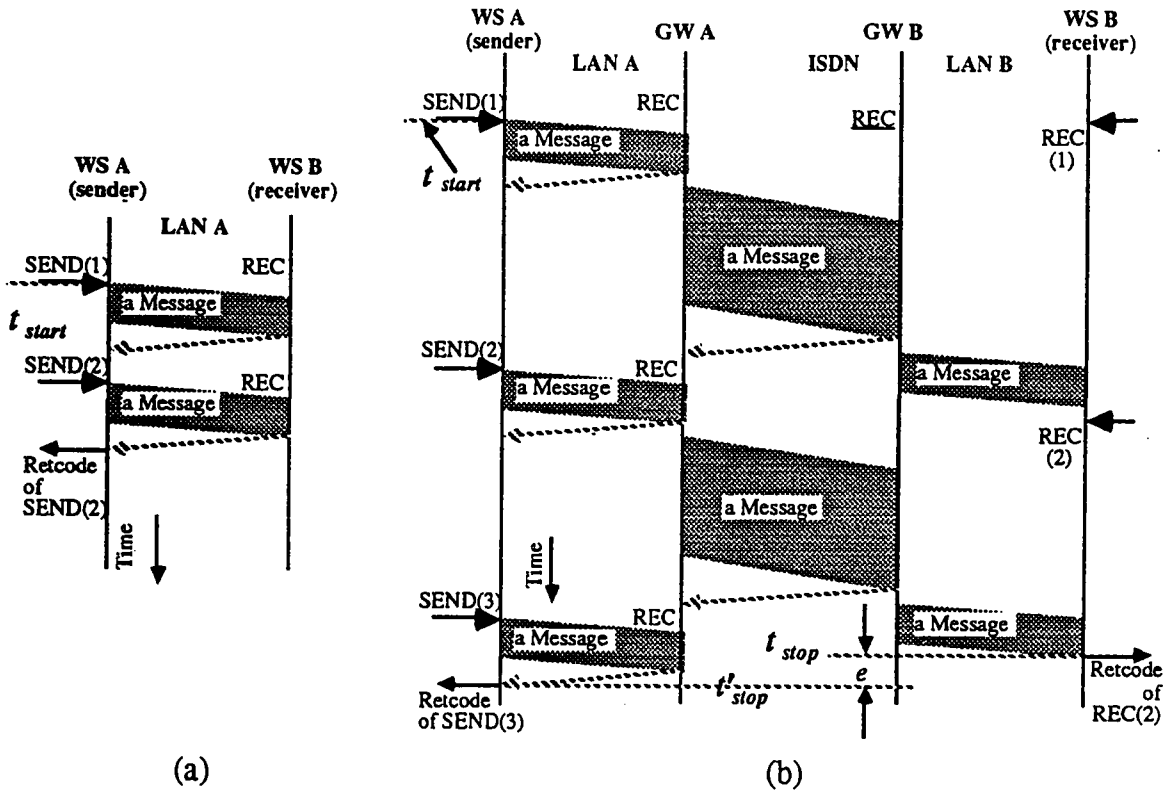


Figure 6.4: Measurement of data transmission time

(a) In an intra-LAN environment (b) In an internetwork environment

not necessary for the sending station to wait until the previous Message has reached the receiving station. Timing diagrams of both intra LAN and inter LAN data transmission are shown in Figure 6.4 (a) and (b). The end-to-end transmission time T contains the following components.

1. Messages processing and transmission time. This elapsed time includes the time which is spent processing protocols under the NetBIOS interface and the time for transmission of packets and acknowledgements on LANs and ISDN.

2. Processing time of applications in stations and in gateways. The elapsed time in an end-station is dependent on the type of application. In the disk-to-disk test program, for example, it contains the disk access time. In the memory-to-memory test program, the software is implemented efficiently so that the processing time in the end-stations can be neglected. In the gateway, the application-layer software is the gateway program. The processing time includes the polling time and the time to relay data from one network to

the other. The first part of the time will vary in terms of the number of active sessions. The second part is defined as the time from getting a return code of a RECEIVE command to issuing a SEND command to the other adapter in the gateway (refer to Figure 6.5). This time is not so significant. It takes about 700 clock periods. If a clock period is 0.1 μs (10 MHz), it will take about 70 μs for a Message relaying from one side of the gateway to the other side. The processing time in network adapters has been taken into account in the messages transmission time.

3. Transmission delay in LANs and in ISDN. This component is dependent on the length of links and the intranetwork and internetwork traffic. We have a back-to-back connection on ISDN and a short link on LANs, therefore, the propagation delay, which is dependent on the length of the link, can be neglected. These components are not going to be measured individually. The end-to-end transmission speed is most interesting in our experiment.

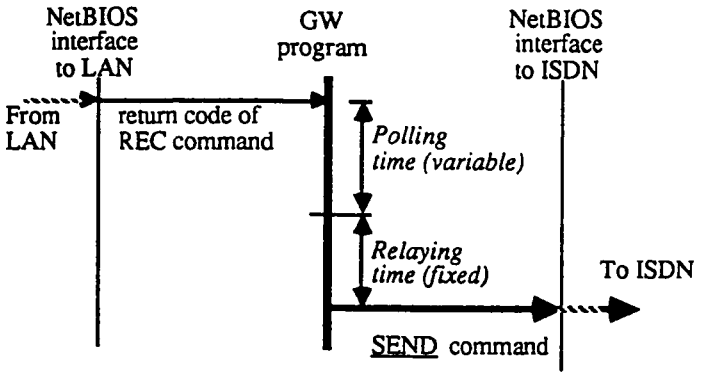


Figure 6.5: Data-relaying time in the gateway

We measure the system with a real-time clock of a PC (personal computer). Otherwise, a special timer has to be configured. However, the measurement of session establishment and termination cannot be made, since the resolution of the clock is only 20 ms. Compared with this resolution, the time to open and close a connection is too short to be measured.

The stopping time of data transmission t_{stop} obtained from the receiving station is not correlated with the starting time t_{start} from the sending station, since clocks of two communicating stations are not synchronized. Thus, the time we get at the receiver is meaningless. All the measurements have to be carried on at the sender. The time t'_{stop} is used instead of t_{stop}. This will bring a measuring error e (refer to Figure 6.4(a) and (b)). Suppose, we want to measure the transmission time of two Messages. In a LAN environment, the time t'_{stop} is obtained after sending the second Message completely (refer to Figure 6.4(a)). However,

in order to minimize the error in the internetwork environment, t'_{stop} is obtained after one more Message (third Message in Figure 6.4 (b)) is sent rather than the last Message (i.e., the second Message). Since the transmission speed in a LAN is much faster than that in an ISDN channel, this arrangement will bring less measuring error. The reduction of the measuring error will also be achieved by transmitting more Messages. With these considerations, the error will be minimized in both environments.

6.3 Tests and Experimental Results

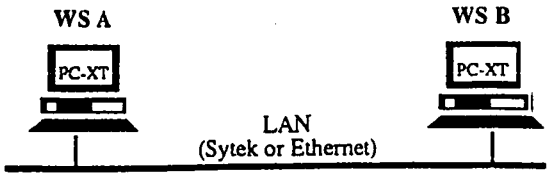
6.3.1 Performance-dependent Factors

The network conditions and the overhead introduced by communications software significantly impacts the performance of most distributed applications. In our experiment, we consider the factors that may affect the performance of an internetwork communication based on our experimental internetworking system:

- *Granularity of communication.* In our case, this term means the size of a session-layer data block (a Message size). Obviously, the more Messages a file is divided, the more the information exchanges on the NetBIOS interface.
- *Background traffic.* For the data transfer on a particular end-to-end link, the background traffic means data transfers on other active end-to-end links. The transmission speed is dependent on the number of active links in one or more subnetworks. We can consider the influence of the intranetwork background traffic and the internetwork background traffic.
- *Traffic at end-station.* In this case, an end-station either receives an incoming data flow or sends an outgoing data flow.

6.3.2 Without background Traffic

The performance of the end-to-end transmission is considered on the condition of no background traffic within a network or between networks so that the maximum data transmission rate can be achieved. Under this circumstance, the end-to-end transmission time with respect to the Message size is tested. Four network configurations are examined, the first two cases are the intranetwork tests, and the last two cases are the internetwork tests.



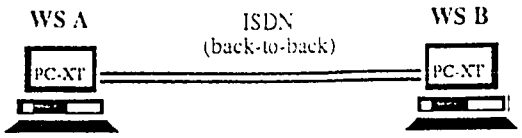
File Size	Message Size	# of Messages	ETHERNET	SYTEK
			WS A--WS B	WS A--WS B
0.5 MB	1 kB	512	82 s *	22 s*
	2 kB	256	47 s *	16 s*
	4 kB	128	19 s	10.3 s
	8 kB	64	14 s	7.6 s
	16 kB	32	14 s	5.6 s
	32 kB	16	12 s	5 s
	64 kB	8	11.5 s	4.5 s

* WS B clock: 5MHz, otherwise 10MHz

Figure 6.6: Intra LAN tests without background traffic

Case A: Within one LAN (shown in Figure 6.6), data transmission on the Ethernet LAN and the Sytek LAN are examined with different Message sizes. Figure 6.6 depicts the transmission time of 512k bytes data on both LANs. The results indicate that the smaller the Message size, the slower the end-to-end transmission in both networks. Also the results shows that the processing time on the Ethernet LAN (max. 10 Mbps channel capacity) is slower than that on the Sytek LAN (max. 2 Mbps channel capacity). This is because the Ethernet card does not have an on-board processor. All protocols above the MAC layer and the applications are processed by the host processor. On the other hand, the Sytek adapter uses an on-board microprocessor to speed up the protocol processing. With the NetBIOS interface, the maximum end-to-end throughput with 64k-byte Message size is 0.9/2= 45 % on the Sytek LAN, and 0.356/10 = 3.56 % on the Ethernet LAN. In the Ethernet LAN environments, the end-to-end transmission rate can be improved by a faster processor.

Case B: Within an ISDN (shown in Figure 6.7), data are transmitted between two back-to-back connected ISDN stations. In the same way as the gateway configuration, one station is in the TE mode, the other one should be in NT mode. Using the same test program, we get the results shown in Figure 6.7. The maximum throughput of an ISDN link on a B channel can reach 62 % of the channel capacity of 64 kbps. As shown in the experiment, the Message size does not affect the transmission speed significantly. Also, the data transmission



File Size	Message Size	# of Messages	B channel
			WS A--WS B
0.5 MB	1 kB	512	115.0 s *
	2 kB	256	104.1 s *
	4 kB	128	103.2 s
	8 kB	64	102.4 s
	16 kB	32	100.5 s
	32 kB	16	97.2 s
	64 kB	8	90.4 s

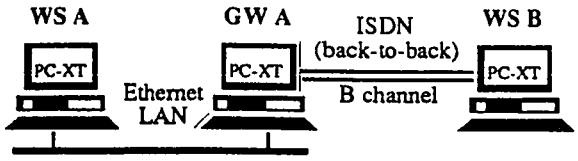
* WS A and WS B clocks: 5MHz, otherwise 10MHz

Figure 6.7: Data transmission within ISDN without background traffic

on the D channel packet-switching link has been measured. The maximum throughput on the D channel can reach 90 % of 16 kbps. These results indicate that the throughput on the B channel or on D channel has exceeded 50 %. If more than one simultaneous transmissions are on one of the ISDN channels, the throughput of an end-to-end transmission will be less than the maximum values which have been obtained above.

Case C: The cases A and B are in a single network environment. Now, tests between networks are considered. Between a LAN and an ISDN (shown in Figure 6.8), data are transmitted through one LAN/ISDN gateway we designed. The gateway is designed for stop-and-wait operation on the session layer. A Message is sent from the LAN station just after the gateway has sent the previous Message to the ISDN station. Since only two peer commands are polled, the processing time of the gateway program is so small that it can be neglected. Thus, the end-to-end data transmission time will be the transmission time of A plus the one of case B. The results listed in Figure 6.8 shows that the data transmission time from a LAN station to an ISDN station is the sum of two times from case A and case B. The transmission at the ISDN side of the gateway is the bottleneck of the end-to-end data transmission.

Case D: Between a Sytek LAN station and an Ethernet LAN station through ISDN B channel (shown in Figure 6.9), data are transmitted through two LAN/ISDN gateways. In this case, the processing time of the gateway program in two gateways still can be neglected. Although the gateway is designed in stop-and-wait operation on the session layer, as we



File Size	Message Size	# of Messages	Via B channel
			WS A--WS B
0.5 MB	1 kB	512	187 s *
	2 kB	256	155 s *
	4 kB	128	120 s
	8 kB	64	116 s
	16 kB	32	113 s
	32 kB	16	110 s
	64 kB	8	103 s

* GW A and WS B clocks: 5MHz, otherwise 10MHz

Figure 6.8: Data transmission between LAN and ISDN without background traffic

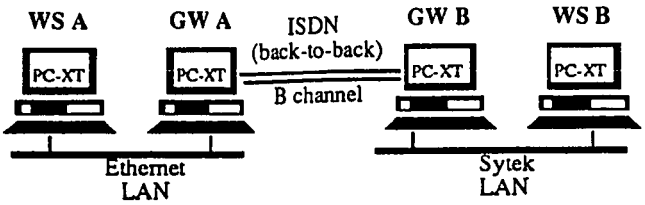
analyzed in the last section, data transfers on two LANs can be concurrent if the end-to-end acknowledgement of a Message transfer is not applied. The transmission time can be reduced. The results listed in Figure 6.9 shows that the transmission time is almost the same as that in the case C.

Concerning the Message size, we can summarize from the test results of the four cases above:

- It is better to choose large Message sizes for the data transmission within a LAN. Thus, the efficiency of the data transmission will be higher.
- In the cases of a LAN-to-ISDN communication and a LAN-to-LAN communication through ISDN, the Message size is not an important factor to affect the transmission speed. A smaller message size, such as 8k bytes may be chosen so that the storage requirement in gateways may be reduced. In other words, more buffers can be allocated in a limited size memory.

6.3.3 With background Traffic

We have examined the cases in which data transmission is on an "empty" interconnected network. We now test two other cases with background traffic.



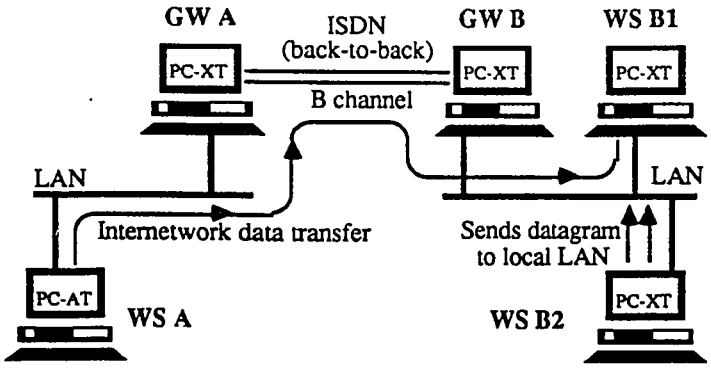
File Size	Message Size	# of Messages	Via B channel
			WS A--WS B
0.5 MB	1 kB	512	192 s *
	2 kB	256	157 s *
	4 kB	128	120 s
	8 kB	64	116 s
	16 kB	32	113 s
	32 kB	16	110 s
	64 kB	8	103 s

* GW A and GW B clocks: 5MHz, otherwise 10MHz

Figure 6.9: Data transmission between LANs through ISDN without background traffic

Case E: Background Traffic within a LAN. In this case, a LAN station which is not involved in our end-to-end data transmission sends packets to its attached LAN continuously as shown in Figure 6.10. Since no more LAN station is available, the data transmission within LAN (as background traffic) is with the datagram support of the NetBIOS. No packet acknowledgement is required in datagram mode. These packets are isolated in a single LAN since the relay of a datagram is not implemented in our prototype gateway. The results listed in Figure 6.10 shows that the intra-LAN background traffic which comes from one station is too light to affect the end-to-end internetwork trānsmission. It is not surprising because a LAN provides a high network capacity.

Case F: Background Traffic across the Gateways. We put the background traffic in the interconnected network by setting up another internetwork session besides the end-to-end session under test (refer to Figure 6.11). Therefore, data transmissions on two parallel sessions between LANs are realized. Both sessions are on the same ISDN B channel (the B1 channel). The transmission time on this configuration is listed in Figure 6.11. It can be found that the transmission time with this background traffic is two times of that without background traffic. The ISDN link (including an ISDN channel and the ISDN protocols up to the Session layer) limits the end-to-end



Message Size	Background Datagram Size	Transmission Time
	At local LAN	WS A-WS B1
5k x 5 = 25k bytes	No	10.6 s
	512 bytes	10.6 s
Clocks: XT: 5MHz, AT: 8 MHz		

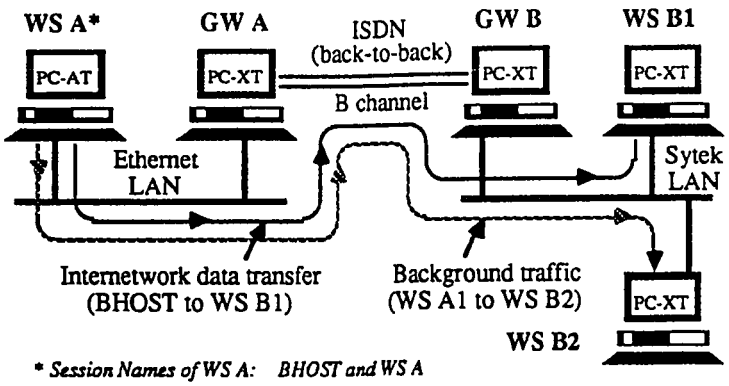
Figure 6.10: Data transmission with intra-LAN background traffic

transmission speed. Therefore, multi-session internetwork data transmission will bring more delay in ISDN because of the limitation of the ISDN bandwidth.

6.4 Summary

In order to assess the performance of the prototype gateway, we have done the data transfer tests on an internetworking system in which the ISDN gateways are back-to-back connected. The disk-to-disk test program is used for the emulation of a file transfer between LANs through an ISDN channel, and the memory-to-memory test program is used for the performance evaluation to the system. Three factors which may affect the performance of the end-to-end data transmission have been identified. Since the intranetwork traffic and the internetwork traffic vary with different offered loads, it is not possible to test all the network configurations. Therefore, we test the end-to-end data transmission time in six different cases.

In the no background traffic cases, we find that the internetwork transmission time (case C and case D) is close to the transmission time of two ISDN stations (case B). We also focused our attention on the effect of the Message size. It was found that the



Message Size	Background Message Size	Transmission Time
	BHOST-WS B2	WS A-WS B1
0.5k x 128 = 64k bytes	No	34.4 s
	1k bytes	68.3 s
	2k bytes	68.2 s
	4k bytes	68.2 s
Clocks: XT: 5MHz, AT: 8 MHz		

Figure 6.11: Data transmission with inter-LAN background traffic

Message size does not impact the performance significantly. In this case, the small Message size can be applied to save buffer space in the gateway.

In the cases with background traffic, only two configurations were tested. In case E, the results show that intra-LAN background traffic from one LAN station does not change the internetwork transmission time any more. If more background traffic exists on LAN, the internetwork data transfer will be slowed down. If the internetwork background traffic is applied (case F), the results show that the internetwork data transmission will be greatly slowed down. Apparently, the ISDN channel rather than the Message size is the bottleneck for the data transfer on the internetworking system.

Chapter 7

Concluding Remarks and Suggestions for Further Study

7.1 Concluding Remarks

The ISDN is an internationally specified access standard for the future telecommunication infrastructure. It will carry all services in one single network. At present, the ISDN services make it possible to interconnect existing local data networks through ISDN packet-switched or circuit-switched channels. In our project, a LAN-ISDN gateway on the Session layer has been developed. By using this gateway, a LAN station can communicate with another station sitting in a remote LAN through an ISDN channel and with an ISDN station as well. The interconnection of the voice switching equipments, such as PBXs is not considered in this project.

In this thesis, we first reviewed existing single networks because the services of internetworking devices between these networks are dependent on the characteristics of the networks which are interconnected. Then, we studied the internetworking requirements and issues of designing an internetworking unit (device). It can be implemented in several ways as we discussed in Chapter 3, such as a bridge, a router or a gateway. The main purpose in our project, however, was to establish a link between LANs via ISDN. In order to take advantage of the services of the LAN which has already been in use and the services of the ISDN adapters on hand, an application-layer gateway interfaced with the NetBIOS has been chosen to be the internetworking device between LAN and ISDN.

To define the services of the gateway, specifications with state machine notations were

presented on mapping connection-oriented services from one network to the other and on mapping connectionless services between LANs through a connection-oriented ISDN channel. In the design and implementation phases, our attention was focused on the mapping of the connection-oriented services provided by both LAN and ISDN. Several aspects had to be emphasized, such as session establishment, session termination, end-to-end session acknowledgement, data transfer on a multiple-session logical link, error processing and multi-task handling in the gateway. More than one solution were presented in some aspects to allow the expansion of gateway services according to network conditions and users (or applications) requirements. The limitation of the NetBIOS interface was also considered in the gateway design. A prototype of interconnected networks was configured, which include a Sytek LAN, an Ethernet LAN and ISDN. Since the ISDN public service is not yet available in Canada, a back-to-back connection was used instead; this is not expected to change the results aside from the transmission time and the switching delay.

Finally, we developed two test programs to emulate the application of end stations, one to verify the file transfer across several networks (disk-to-disk data transfer) and the other to test the gateway performance (memory-to-memory data transfer). For the disk-to-disk data transfer verification, an executable file was sent from one LAN to the other. For the memory-to-memory data transfer, the end-to-end transmission speed was tested in different network conditions. From these results, we can find what kinds of applications this implementation was best suited for.

From the design and implementation of the interconnected system, we have the following concluding remarks. The gateway system implemented in our project allows a LAN resource to be accessed by an ISDN computer which is simply connected to the ISDN by a common telephone cable and be shared by another LAN, which is located far from the resource. At the Session layer, both LAN and ISDN are treated as homogeneous networks. This simplifies the development of the gateway program. Complicated protocol translation can be avoided. The gateway uses a unique flat name to route the call to the right destination so that no extra routing algorithm is required. However, the number of simultaneous internetwork calls is limited by the NetBIOS name space.

Concerning the performance, we find that data transfer is reliable on established sessions across the gateway even if the end-to-end acknowledgement is not applied. From the test results, the end-to-end transmission speed is limited by the ISDN channel

capacity and the protocol processing time. Since the transmission speed is not able to exceed the channel capacity, the protocol processing upper to application layer in the gateway is acceptable in this case. However, for the interconnection of high speed networks, processing time of the higher layer protocols will be the internetworking bottleneck.

The ISDN provides a cheap communication link because of the lower speed. Leasing such a line is as easy as having a telephone line at home or office. The test results show that it is acceptable to transmit small or medium size data files in this internetwork system. However, if a large data file, such as a digital image file in a range of several Megabytes, is transmitted through an ISDN channel, it will take longer time to complete the job.

7.2 Further Research Suggestions

As we mentioned above, this application-layer gateway is implemented on the NetBIOS interface. Therefore, any type of network with the NetBIOS interface can be interconnected by this gateway. Besides interconnecting two or more IEEE 802.3 type of LANs, this gateway can interconnect different LANs through ISDN, such as CMSA/CD LAN, another token ring LAN, or any network. The only requirement is that all networks interconnected by the gateway should support the NetBIOS interface. The gateway is designed and implemented in an ISDN basic rate access environment. Due to the similarity of ISDN BRA (basic rate access) and ISDN PRA (primary rate access), this solution of internetworking can be expanded to the ISDN PRA environments. In our prototype of the gateway system, we map the connection-oriented services which have existed in both LAN and ISDN. If the LANs which are interconnected by the ISDN need the datagram service, the datagram can be relayed through the existing ISDN services. The implementation of such conversion will not be too complicated. However, the connectionless service will not be used by ISDN stations directly, or otherwise the ISDN stations must support the datagram service. Another possible expansion is to configure a multi-port gateway to interconnect both data network and voice network through ISDN channels. In the ISDN side of this gateway, separate channels for voice and data are provided. This expansion will merge a local data network and a local voice network into an wide integrated network.

Bibliography

[ANSI86] American Natioanl Standarda Institute, *FDDI Token Ring Media Access Control (MAC): Draft Proposed American National Standard*, Feb. 1986.

[BACK88] F. Backes, "Transparent Bridges for Interconnection of IEEE 802 LANs", *IEEE Network*, Vol. 2, No.1, pp. 5-9, Jan. 1988.

[BERN85] J. A. Berntsen, J. R. Davin, D. A. Pitt, "MAC Layer Interconnection of IEEE 802 Local Area Networks", *Computer Networks and ISDN systems*, Vol. 10, pp. 259-273, 1985.

[BHAR81] K. Bharath-Kumar, "Some issues in Homogeneous Network Interconnection", Research Report RC-8739, IBM Research Division, San Jose, CA., 1981

[BURG89] F. M. Burg, N.Di Iorio, "Networking of Networks: Internetworking According to OSI", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-7, NO.7, pp. 1131-1142, Sept. 1989.

[CALL83] R. Callon, "Internetwork Protocol", *Proceedings of IEEE*, Vol. 71, No.12, Ced. 1983.

[CERF78] V. G. Cerf, P. T. Kirstein, "Issues in Packet-Network Interconnection", *Proceedings of IEEE*, Vol. 6, No.11, pp. 1386-1408, Nov. 1978.

[CCITT80A] CCITT Revised Recommendation X.75, "Terminal and Transit Control Procedures and Data Transfer System on International Circuits Between Packet Switched Data Networks", COM VII-No 441 as amended and approved by Study Group VII, February 1980.

[CCITT80B] CCITT Revised Recommendation X.121, "International Numbering Plan for Public Data Networks", as amended and approved by Study Group VII, February 1980.

[CCITT84] CCITT Recommendation I-series Recommandation, "Integrated Services Digital Network (ISDN)", Red Book, Vol. III - Fascicle III.5, Oct. 1988.

[COLE87] R. Cole, "A Methord for Interconnecting Heterogeneous Computer Networks", *Software - Practice and Experience*, Vol. 17(6), pp. 387-397, June 1987.

[COME88] D. Comer, *Internetworking with TCP/IP, Principles, Protocols, and Architecture*, Prentice Hall, 1988.

[DALE87A] "IBM PC ISDN "S" Co-processor Adapter", DALE, GESEK, McWILLIAMS & SHERIDAN, INC., 1987

[DALE87B] "DGM&S Co-processor Software Development and Test Environment", DALE, GESEK, McWILLIAMS & SHERIDAN, INC., 1987

[DALE88] "ISP188/ISDN Basic Access Product, Installation/Reference handbook", DALE, GESEK, McWILLIAMS & SHERIDAN, INC., 1988

[DANT82] A. A. S. Danthine, "Network Interconnection", *Local Computer Networks*, edited by P. C. Pavasio, G. Hopkins, N. Naffah, pp. 289-308, North-Holland, 1982.

[DANT90] A. Danthine, *et al*, "Access Rate Measurement in the BWN Environment", *High Speed Local Area Networks, II*, pp. 89-115, IFIP, 1990.

[DAY83] J. D. Day, H. Zimmermann, "The OSI Reference Model", *Proceedings of the IEEE*, Vol. 71, No.12, pp. 1334-1340, Dec. 1983.

[DEAS86] R. J. Deasington, *X.25 Explained: protocols for packet switching networks*, 2nd. ed., Ellis Horwood Ltd., 1986.

[DECI86] M. Decina, E. L. cace, "CCITT Recommendations on the ISDN: A Review", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 3, pp.320-325, May, 1986.

[DONO90] Dono Van-Mierop, "Extending Ethernet/802.3 over FDDI Using the FX 8000", *High Speed Local Area Networks,II*, pp. 259-266, IFIP, 1990.

[ELSA86] E. Elsam, "The Defense Data Network Hits Its Stride", *Telecommunications*, May 1986.

[ENG88] S. T. Eng, L. A. Bergman, "Gigabit Optic Computer Networks: Technology and System Considerations", EFOC/LAN, pp. 426-429 1988.

[GEOR89] N. D. Georganas, M. Goldberg, L. Orozco-Barbosa and J. Mastronardi, "A Multimedia Communications System for Medical Applications", *Proc. ICC'89*, Boston, USA, June 1989.

[GIEN79] M. Gien, H. Zimmermann, "Design Principles for Network Interconnection", *IEEE-ACM proceedings of Sixth Data Communications Symposium*, pp. 109-119, 1979.

[GLAS89] B. Glass, 'Understanding NetBIOS", *BYTE*, pp. 301-306, Jan. 1989.

[GREE88] P. E. Green, Jr., "Network Interconnection and Protocol Conversion", *IEEE Press Selected Reprinted Series*, IEEE Press, New York, 1988.

[HATL87] D. J. Hatley, *Strategies for real-time system specification*, Dorset House Publishing Co.,Inc., New York, 1987.

[HAWE87] B. Hawe, A. Kirby, B. Stewart, "Transparent Interconnection of Local Networks with Bridges", *Advances in Local Area Networks*, edited by K. Kummerle *et al*, pp. 482-495, IEEE Press, 1987.

[HENS88] J. Henshall, S. Show, OSI Explained, Jhon Wiley & Sons, 1988.

[HU86] W. Hu, "Design and Implementation of a Gateway between Two Electronic Mail Systems", *IEEE Communication Magazine*, Vol. 24, No.6, pp18-23, June 1986.

[IBM84A] "Technical reference - PC Network",*PC Hardware Ref. Library 6322916* IBM, Boca Raton, FL, 1984.

[IBM84B] "IBM PC Network Tanslator Unit", *Ref. Library 6322920*, IBM, Boca Raton, 1984.

[IBM86] "Token-Ring Network/PC Network Interconnect User's Guide", 6466914, IBM, February, 1986.

[IBM87] "IBM Token Ring Network Architecture Reference" 2nd ed. IBM, August, 1987.

[IEEE802.1] IEEE Draft standard P802.1; Section 6 - Networking, Revision - Feb. 1984, Reprinted in *Tutorial on Local Area Network Technology, 3rd* by W. Stallings, IEEE Computer Society Press, 1988.

[IEEE88] Draft Standard IEEE P802.6, Distributed Queue Dual Buses Metropolitan Area Network (DQDB MAN), "Media Access Control (MAC) and Physical Layer Protocol Documents", Draft D.6, Nov. 1988.

[ISO83] "Data processing-Open system interconnection-Basic reference model", Standard ISO-7498, ISO, Geneva, Switzerland, 1983.

[ISO84] "Information processing systems - Open systems interconnection - Connection oriented transport protocol specification", Draft international standard ISO/DIS 8073, ISO/TC 97, 1984.

[JANS87] P. A. Janson, A. A. R. Cockburn, "Adding Transparent Internetworking to a LAN Application Interface", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-5, NO.9, pp. 1471-1479, Dec. 1987. Reprinted in [Gree88], pp. 425-433.

[JENS90] M. N. Jensen, M. Skov, "VLSI-architectures Implementing Lower Layer Protocols in Very High Data Rate", *High Speed Local Area Networks, II*, pp. 187-205, IFIP, 1990.

[JOSH86] S. P. Joshi, "High-Performance Networks: A Focus on the Fiber Distributed Data Interface (FDDI) Standard", *IEEE Micro*, pp. 8-14, June 1986.

[KANO86] S. Kano, "Layers 2 and 3 ISDN Recommendations", *IEEE JSAC*, vol SAC-4, no. 3, pp. 355-359, May 1986.

[KELL90] P. Kelly, "Connecting LANs with Bridges", *Telecommunications*, pp. 31-38, June 1990.

[KLEE87] M. Kleeman,*et al.*, *PC LAN Primer*, Howard W. Sams & Co., 1987.

[KNOW87] T. Knowles, J. Larmouth, K. G. Knightson, *Standards for Open Systems Interconnection*, BSP Professional Books, 1987.

[LACH88] "The Lanchman ISDN-LAN Bridge features TCP/IP and Network File System (NFS) capabilities over ISDN", *ISDN-LAN Application Note*, Lachman Associates, Inc., May 1988.

[LAI89] W. S. Lai, "Frame Relay Service: An Overview", *Proceeding s, IEEE INFOCOM 89*, Ottawa, Canada, April 1989.

[LAMO89] J. Lamont, J. Doak, M. Hui, "LAN Interconnection via Frame Relaying", *Proceeding s, IEEE INFOCOM 89*, Ottawa, Canada, pp. 686-690, April 1989.

[LING87] N. Linge, *et al*, "A Bridge Protocol for Creating a Spanning Tree Topology within an IEEE 802 Extended LAN Environment", *Computer Networks and ISDN Systems*, Vol. 13, pp. 323-332, 1987.

[MCCO88] J. McConnel, *Internetworking Computer Systems - Internetworking Networks and Systems* , Prentice Hall, 1988.

[MICR87] *Microsoft C Optimizing Compiler*, V.5.1, Microsoft Inc., 1987.

[MITE87] "The ISDN Express Card, Designer's manual", Mitel Corperation, 1987.

[MITE88] *LAPD Toolkit Package*, Mitel Corperation, 1988.

[MOLL88] J. F. Mollenauer, "Standards for Metropolitan Area Networks", *IEEE Communications Magazine*, vol. 26, no. 4, pp. 15-19, April 1988.

[NEG88] S.T.Eng, L.A.Bergman, "Gigabit Optic Computer Networks: Technology and System Considerations", EFOC/LAN, 1988

[NEWM88] R. M. Newman, Z. L. Budrikis, J. L. Hullett, "The QPSX MAN", *IEEE Communications Magazine*, vol. 26, no. 4, pp. 20-28, April 1988.

[NORT87A] "ISDN PC Terminal Adapter - User Guide", PO681932, Northern Telecom, 1987.

[NORT87B] "Addendum to PCTA User Guide", Issue 3.1, PO681932, Northern Telecom, 1987.

[NORT87C] "ISDN PC Terminal Adapter - Unpacking and Installation Guide", PO681933, Northern Telecom, 1987.

[NORT87D] "ISDN PCTA - Operating requirements, Installation, and Verification ", PO689829 - Minimum PCR25, Northern Telecom, 1987.

[NORT87E] "Meridian ISDN Digital Telephone T2317", PO673913, Northern Telecom, 1987.

[NORT88] Northern Telecom, "ISDN PC Terminal Adapter Net BIOS - Interface Description",NIS Number D307-1, 1987.

[PERL88] R. Perlman, A. Harvey, G. Varghese, "Choosing the Appropriate ISO Layer for LAN Interconnection", *IEEE Network*, Vol. 2, No.1, pp. 81-86, Jan. 1988.

[PIER72] J. R. PIERCE, "Network for Block Switches of Data", *Bell System Technical Journal*, July/August, 1972.

[PITT87A] D. A. Pitt, "Standaras for the Token Ring", *IEEE Network Magazine*, Vol. 1, pp. 19-22, Jan. 1987.

[PITT87B] D. A. Pitt, K. K. Sy, R. A. Donnan, "Source Routing for Bridged Local Area Networks", *Advances in Local Area Networks*, edited by K. Kummerle *et al*, pp. 517-530, IEEE Press, 1987.

[PITT87C] D. A. Pitt, J. L. Winker, "Table-Free Bridging", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-5, No. 9, pp. 1454-1462, Dec. 1987.

[POST80] J. B. Postel, "Internetwork Protocol Approaches", *IEEE Transactions on Communications*, Vol. COM-28, No.4, pp. 604-611, April 1980.

[ROBE70] L. G. Robert, B. D. Wessler, "Computer Network Development to Achieve Resource Sharing", *Proceedings of SJCC*, pp. 543-549, 1970.

[ROSS86] F. E. Ross, "FDDI - A Tutorial", *IEEE Communications Magazine*, vol 24, no. 5, pp. 10-17, May 1986.

[RYBC80] A. Rybczynski, "X.25 Interface and End-to-End Virtual Circuit Service Characteristics", *IEEE Trans. on Comm.*, vol COM-28, no. 4, pp. 500-509, April 1980.

[SCHN83] N. F. Schneidewind, "Interconnecting Local Networks to Long-Distance Networks", *IEEE Computer*, pp. 15-24, Sept. 1983.

[SCHW87] M. Schwartz, *Telecommunication Networks*, Addison-Wesley Publishing Company, 1987.

[SEIF88] W. M. Seifert, "Bridges and Routers", *IEEE Network*, Vol.2, No.1, pp. 57-64, Jan. 1988.

[SKOV89] M. Skov, "Implementation of Physical and Media Access Protocols for High-Speed Networks", *IEEE Communications magazine*, Vol. 27, No. 6, pp.45-53, June, 1989.

[STAL87A] W. Stallings, *Handbook of computer communications Standards: The OSI Model and OSI-Related Standards, Vol. 1*, Howard W. Sams & Company, Indiana, 1987.

[STAL87B] W. Stallings, *Handbook of computer communications Standards: Local Network Standards, Vol. 2*, Howard W. Sams & Company, Indiana, 1987.

[STAL87C] W. Stallings, *Handbook of computer communications Standards: Department of Defense (DOD) Protocol Standards, Vol. 3*, Howard W. Sams & Company, Indiana, 1987.

[STAL87] W. Stallings, *Local Area Networks, An Introduction, 2nd ed.*, Macmillan Publishing, New York, 1987.

[STAL88] W. Stallings, *Data and Computer Communications, 2nd ed.*, Macmillan Publishing, New York, 1988.

[STAL89] W. Stallings, "When One LAN Is Not Enough", *BYTE*, pp. 293-298, Jan. 1989.

[STEW84] B. Stewart, B. Hawe, A. Kirby, "Local Area Network Connection", *Telecommunications*, pp. 54-66, April, 1984

[STIE85] M. Stieglitz, "IBM provides Industry with a Versatile Local Network Standard", *Data Communication*, pp. 195-206, June 1985.

[SY87] K. K. Sy, *et al.*, "OSI-SNA interconnections", *IBM systems Journal*, Vol. 26, No.2, pp. 157-173, 1987.

[SYTE86] Sytek, "Sytek 6120 Network Adaptor Card Technical Reference Manual", Publication Number R1015.1/1-01A, 1986.

[TANE88] A. S. Tanenbaum, *Computer Networks*, Prentice Hall Inc., New Jersey, 1988.

[UNSO82] M. S. Unsoy, "X.75 Internetworking of Datapac with Other Packet Switched Networks", *Journal of telecommunication Networks*, Vol.1, No. 3, fall 1982.

[VALL89] Richard Vallée, Luis Orozco-Barbosa and N. D. Georganas, "Interconnection of Token Rings by IEEE 802.6 Metropolitan Area Networks", *INFOCOM'89*, Ottawa, Canada, pp. 934-942, April 1989.

[VOEL86] J. Voelcker, "Helping Computers Communicate", IEEE Spectrum, pp. 61-77, 1986.

[WEST87] "EtherCard Plus, user Instrallation Guide", Western Digital Corperation, 1987.

[WILE] A. J. Wiley, "Applying International Standards to ISDN-LAN Interconnection", University of Lancaster, UK.

[ZHAN88] L. Zhang, "Comparison of Two Bridge Routing Approaches", *IEEE Network*, Vol. 2, No.1, pp. 44-48, Jan. 1988.

[ZOLI85] K. O. Zoline, W. P. Lidinsky, "An Approach for Interconnecting SNA and XNS Networks", *IEEE-ACM Proceedings of the Ninth Data Communications Symposium*, pp 184-198, 1985.

Appendix A

Gateway Program

A.1 GW_MAIN.C

```
/*******************************************************/
/*                                                     */
/* GW_MAIN.C--main entrance of the gateway program.    */
/*      --The main routine checks the initialization of the */
/*        gateway. If the gateway has been initialized, the */
/*        gateway  program is allowed to execute.      */
/*                                                     */
/*      --It is a part of GATEWAY.EXE.                 */
/*      --Compiled with MS C v.5.0.                    */
/*                                                     */
/*******************************************************/

#include <conio.h>
#include <stdio.h>
#include "netbios.h"

short int       init_flag = NON_INIT;  /* set init_flag to non init status */

void main (void)
{
    char        key_input;
    while (key_input!=ESC)
    {
        printf ("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
        printf ("\n\n\t< 1 > ========== INITIALIZATION.");
        printf ("\n\n\t< 2 > ========== RUNING GATEWAY PROGRAM.");
        printf ("\n\n\t<Esc> ========== EXIT to DOS.");
        printf ("\n\n\n\n\n\n\n");
repeat: printf ("\n\tYOUR  SELECTION ===>> ");
        key_input = getche();
        switch (key_input)
        {
            case INITIAL:
                init_gateway (); /* initialize the gateway */
                init_flag=INIT;  /* set flag to be INIT */
```

```
            break;
        case RUN_GW:
        {
            if (init_flag==INIT)
            {   /* the gateway has been initialized */
                run_gateway (); /* gateway program running */
                break;
            }
            else
            {
                printf ("\n\n\tThe program has not been initialized.");
                printf ("\n\n\tPlease enter < 1 > to initialize first!");
                goto repeat; /* return to main menu */
            }
        }
        case ESC: break;    /* exit from MAIN program to DOS. */
        default:    break;
    }
}
    /* if "Esc" key is hit , program stopped */
exit(0); /* exit to DOS */
}
```

A.2 GW_INIT.C

```
/*************************************************/
/*                                               */
/*  GW_INIT.C--initialization program to add or delete  */
/*        a name in LANA Name Table, and to initialize  */
/*        the Session Name Table.                */
/*  -- It is the part of GATEWAY.EXE.            */
/*                                               */
/*************************************************/

#include <conio.h>
#include <stdio.h>
#include "netbios.h"

char    lan_name[16][16];
char    temp_name[16];

void init_gateway(void)
{
    extern  short int   init_flag;
    char                in_key;
    while (in_key!=ESC)
    {
        printf ("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\t<A> or <a>.....Adding or Deleting names in
                adapter(Sytek or Ethernet)");
        printf ("\n\n\t<B> or <b>.....Adding or deleting session names in gateway.");
        printf ("\n\n\t<Esc>.....Exit from name initializing program.");
        printf ("\n\n\n\n\n\n\n");
        printf ("\n\t-----Enter ---->> ");
        in_key=getche( );
        printf ("\n\n");
        switch (in_key)
        {
            case ADD_DEL_LAN_NAME_A:
            case ADD_DEL_LAN_NAME_AA:
                lan_name_support ();
                break;
            case ADD_DEL_SESS_NAME_B:
            case ADD_DEL_SESS_NAME_BB:
                session_name_support ();
                break;
            case ESC:  break;    /* exit from init_gateway( ) routine */
            default:   break;
        }
    }
}


void lan_name_support ( void) /* add and delete a lan adapter name */
{
    short int     i; /* used in FOR loop */
```

```
    char        lana_num=ZERO,  kybrd = ZERO;
    short int   lana,       name_index =1 ;
    while((lana_num & 0x00fe) != ASCII_ZERO)
    {
        printf("\n\tSelect 'lana number' of LAN adapter. It should");
        printf("\n\tdiffer from that of ISDN PCTA. < 0 > or < 1 >: ENTER ==>> ");
        scanf ("%c", &lana_num);
    }
    lana=lana_num&0x0001;
    add_name_ncb.lana_num=lana; /* set the lana field in the NCB */
    del_name_ncb.lana_num=lana;
        /* print a name table */
prt1:   printf ("\n\n\t (The LAN adapter number is------ %d)", lana);
    printf ("\n\n\t ---------- A D A P T E R  N A M E  T A B L E ----------\n");
    printf ("\t|\t                  |\t\t\t   |");
    printf ("\n\t|\tRemote  Names     |\tRemote  Names    |");
    printf ("\n\t|\t                  |\t\t\t   |\n");
    while( name_index<9)
    {
        printf("\t| %d\t%-16s     | %d\t%-16s   |\n",name_index,
            lan_name[name_index -1],name_index+8,lan_name[name_index+7]);
        if (name_index!=9)
        name_index++;
    }
    printf ("\t|\t                  |\t\t\t   |\n");
    printf ("\t ----------------------------------------------------------\n");
    name_index=1;

        /* allow to change the name */
    while (kybrd!=ESC)      /* loop,if input is not <Esc> */
    {
        printf ("\n\tPress key <Esc> to exit from this procedue, or ");
        printf ("\n\tPress key < r > to modify REMOTE LAN's name again, or");
        printf ("\n\tPress key < c > to check the name table.\n");
        printf ("\t--->> ");
        kybrd=getche();
        switch (kybrd)
        {
            caseESC: break;
            case MOD_NAME_R:
            case MOD_NAME_RR:
            {
                short int   get_num;
                char        c;
                printf ("\n\tWhich REMOTE NAME need to be modify? Enter number....");
                if (scanf ("%d",&get_num)==0)
                {
                    scanf ("%s",&c);
                    printf ("\n\t This is an illegal number, try again!");
                    break;
                }
                else
                if (get_num < 1 || get_num > 16)
```

```
                {
                    printf ("\n\t This is an illegal number, try again!");
                    break;
                }
                printf ("\n     OLD NAME %2d.....%s",get_num,lan_name[get_num -1]);
                printf ("\nEnter NEW NAME %2d (Enter 0 to delete OLD NAME)...",get_num);

                copy_old_name (get_num);
                scanf ("%s",lan_name[get_num-1]);
                if (lan_name[get_num-1][0]==ASCII_ZERO)
                {
                    printf ("\n\n\t*****************************************
                    ************************");
                    printf ("\n\n\tDelete old remote name <%s> in LAN card (Sytek or
                            Ethernet)",temp_name);
                    if (del_lan_name()==OK)
                    {
                        for(i=0;i<16;i++)
                        lan_name[get_num-1][i]=0;
                    }
                    else
                    {
                        for (i=0;i<16;i++)
                        lan_name[get_num-1][i]=temp_name[i];
                    }
                    printf ("\n\n\t*********************************
                    *********************************");
                }
                else
                {
                    printf ("\n\n\n\n\n\n\t*************************************
                    ************************");
                    printf ("\n\n\t\Add remote name <%s> in LAN card (Sytek or
                            Ethernet)",lan_name[get_num-1]);
                    if (add_lan_name(get_num)==ERROR)
                    {
                        for (i=0;i<16;i++)
                        lan_name[get_num-1][i]=0;
                        printf ("\n\n\t>>>>>>>>> This name is aborted.");
                    }
                    printf ("\n\n\t*************************************
                    ************************");
                }
                break;
            }
            case CHK_NAME_C:
            case CHK_NAME_CC: goto prt1;  /* print the name table again */
            default:    break;
        }
    }
}
```

```
int    add_lan_name(short int got_num)
{
       short   int  i,  err=OK;
       for(i=0;i<16;i++)
       {
           if (lan_name[got_num-1][i]==0 ||lan_name[got_num-1][i]=='\n')
           add_name_ncb.local_name[i]=ASCII_SP;
           else
           add_name_ncb.local_name[i]=lan_name[got_num-1][i];
       }
       INT_5C (&add_name_ncb);                    /* issue ADD_NAME command  */
       printf ("\n\t after int 5c the cmd_cplt is %x", add_name_ncb.cmd_cplt);
       while ((add_name_ncb.cmd_cplt & 0x00ff)==CMD_PENDING);
       printf ("\n\n\t after wait a while cmd_cplt is %x", add_name_ncb.cmd_cplt);
       switch (add_name_ncb.cmd_cplt & 0x00ff)
       {
           case GOOD_RETURN: printf ("\n\tADD_NAME OK!");  break;
           case DUP_NAME:       printf ("\n\tOK! NAME had existed in adapter.");  break;
           case NAME_TAB_FULL:
           {
               err=ERROR;
               printf ("\n\n\tName table full. (Rec.: delete some names in name table)");
               break;
           }
           case NAME_USED:
           {
               err=ERROR;
               printf ("\n\n\tName %d is used by another adapter.
                                   (Rec.:specify another name.)", got_num);
               break;
           }
           default: err=ERROR;
           err_process(&add_name_ncb,0);
       }
       return (err);
   }

   void copy_old_name(short int got_num)                    /* copy a existing name */
   {
       short int         i;
       for (i=0;i<16;i++)
       temp_name[i]=lan_name[got_num-1][i];
   }

   int  del_lan_name (void)        /* delete name from adapter name table */
   {
       short int     i, err=OK;
       for (i=0;i<16;i++)
       {
           if(temp_name[i]==0 || temp_name[i]=='\n')
               del_name_ncb.local_name[i]=ASCII_SP;
           else
               del_name_ncb.local_name[i]=temp_name[i];
```

```
    }
    INT_5C (&del_name_ncb);                    /* issue DEL_NAME command */
    while ((del_name_ncb.cmd_cplt & 0x00ff)==CMD_PENDING);
    switch (del_name_ncb.cmd_cplt & 0x00ff)
    {
        case GOOD_RETURN:  printf ("\n\n\tDELETE_NAME OK!");
        break;
        case NAME_NOT_FND: printf ("\n\n\t OK! Name is not found in card.");
        break;
        case NAME_USED:
        {
            printf ("\n\n\t This name can not be deleted because of a active ");
            printf ("\n\t session or a LISTEN command associated with it.");
            err=ERROR;
            break;
        }
        default:
        err=ERROR;
        err_process(&del_name_ncb,0);
    }
    return (err);
}

    /* add and delete a name in the session name table */
void session_name_support (void)
{
    short int    i; /* used in FOR loop */
    char    kybrd =ZERO;
    short int    r_index,l_index;
    while (kybrd!=ESC)        /*  loop, if <Esc> key is not hit */
    {
        l_index=1;
        r_index=1;
prt2:       printf ("\n\n\n\n\n\n\n\n\t ----------S E S S I O N  N A M E  T A B L E ----------\n");
        printf ("\t\t                    \t\t\t\t   |");
        printf ("\n\t\tremote   names   |    local   names      |");
        printf ("\n\t\t                    \t\t\t\t   \n");
        while(r_index<=8 || l_index<=8)
        {
            printf ("\t|r-%2d\t%-21s|l-%2d\t%-20s\n",r_index,lsn_ncb[r_index-1].local_name,
            l_index,lsn_ncb[l_index+7].local_name);
            if (r_index<=8)
            r_index++;
            if (l_index<=8)
            l_index++;
        }
        printf ("\t\t                    \t\t\t\t   \n");
        printf ("\t ----------------------------------------------------\n");
        r_index=1; /* remote name table index */
        l_index=1; /* local name table index */
        while (kybrd!=ESC)
        {
            printf ("\n\n\t\tDo you want modify This NAME table again?\n");
```

```
printf ("\n\tPress key <Esc> to exit from this procedue, or ");
printf ("\n\tPress key < l > to modify LOCAL  LAN's  name again, or");
printf ("\n\tPress key < r > to modify REMOTE LAN's name again, or");
printf ("\n\tPress key < c > to check the session name table.\n");
printf ("\t--->>  ");
kybrd=getche();
switch (kybrd)
{
    case ESC: break;
    case REMOTE_NAME_R:
    case REMOTE_NAME_RR:
    {
        short int    get_num_r;
        char         get_char_r;
        printf ("\n\tWhich NAME of the REMOTE LAN need to be modify? Enter
            number....");
        get_char_r=getche ();
        get_num_r = (get_char&0x00ff)-ASCII_ZERO;
        if (get_num_r < 1 || get_num_r > 8)
        {
            printf ("\n\t This is an illegal number, try again!");
            break;
        }
        printf ("\n   Old NAME%2d.....%s",
                    get_num,lsn_ncb[get_num_r-1].local_name);
        printf ("\nEnter new NAME%2d (Enter 0 to clear this name)...",get_num_r);
        scanf ("%s",lsn_ncb[get_num_r-1].local_name);
            /* ascii zero changed to binary zero */
        if (lsn_ncb[get_num_r-1].local_name[0]==ASCII_ZERO)
        {
            for (i=0;i<16;i++)
            lsn_ncb[get_num_r-1].local_name[i]=0;
        }
        else
        {
            /* the entered remote session name should match one in lan adapter */
            check_rs_name(get_num_r); /* check an entered name */
        }
        break;
    }
    case LOCAL_NAME_L:
    case LOCAL_NAME_LL:
    {   short int    get_num_l;
        char         get_char_l;
        printf ("\n\t Which NAME of the LOCAL LAN need to be modify? Enter
             number....");
        get_char_l = getche ();
        get_num_l = (get_char_l&0x00ff)-ASCII_ZERO;
        if (get_num_l <1 || get_num_l >8 )
        {
            printf ("\n\t This is an illegal number, try again!");
            break;
        }
```

```
                    printf ("\n     Old NAME%2d.....%s",get_num_l,
                                                lsn_ncb[get_num_l+7].local_name);
                    printf ("\nEnter new NAME%2d (Enter 0 to clear this name)...",get_num_l);
                    scanf ("%s",lsn_ncb[get_num_l+7].local_name);
                    if (lsn_ncb[get_num_l+7].local_name[0]==ASCII_ZERO)
                    {
                        for (i=0;i<16;i++)
                        lsn_ncb[get_num_l+7].local_name[i]=0;
                    }
                    break;
                }
                case CHK_NAME_C:
                case CHK_NAME_CC:
                        goto prt2;
                default:    break;
            }
        }
    }
}


void check_rs_name (short int got_num_r)
{
    short int    num_result, i;

        /* compare the entering session name with 16 lan adapter names */
    for(i=0;i<16,i++)
    {
        num_result=strcmp (lsn_ncb[got_num_r-1].local_name, lan_name[i]);
        if (num_result==0)     /* two string are ideltical */
        return;
    }
        /* no name matches */
        /* delete the entered session name */
    for (i=0;i<16;i++)
    lsn_ncb[got_num_r-1].local_name[i]=0;
    printf ("\n\n\n\n\tThe name r-%d is not found in adapter name table,
                                                WRONG NAME!",got_num_r);
    printf ("\n\n\t>>>>>>>> Please re-enter  name r-%d.  <<<<<<<",got_num_r);

}
```

A.3 GW_RUN.C

```
/************************************************/
/*                                              */
/*  GW_RUN.C --process the gateway function.    */
/*                                              */
/*     --It is a part of GATEWAY.EXE.           */
/*                                              */
/************************************************/

#include <conio.h>
#include <stdio.h>
#include "netbios.h"

extern struct ncb lsn_ncb[], rec_ncb[], send_ncb[], call_ncb, add_name_ncb;

/* This is a real-time program of session layer gateway with NetBIOS commandes. */

char   buff_1  [0xffff];                    /* for odd- number sessions */
char    buff_2 [0xffff];                    /* for even-number sessions */

void run_gateway (void)
{
    short int         num, i;
    printf ("\n\n\n\n\n\n\n\n\n\tRUNNING THE GATEWAY PROGRAM.....,");
    for (num = 0; num < 16; num++)
    {
        if(lsn_ncb[num].local_name[0]!=0)
        {
            lsn_ncb[num].command=LISTEN;
            lsn_ncb[num].remote_name[0]='*';
            for (i=1;i<16;i++)
            lsn_ncb[num].remote_name[i]=ASCII_SP;
            if (num<=7)
            {
                lsn_ncb[num].lana_num=add_name_ncb.lana_num;
                lsn_ncb[num].rto_sto=0x6464;       /* rto=sto = 50 sec in LAN card */
            }
            else
            lsn_ncb[num].lana_num=add_name_ncb.lana_num ^ 0x0001;
                /* change from binary zero to ascii space */
            for (i=0;i<16;i++)
            {
                if (lsn_ncb[num].local_name[i]==0)
                lsn_ncb[num].local_name[i]=ASCII_SP;
            }
            INT_5C(&lsn_ncb[num]);
                /* change from ascii space to binary zero */
            for (i=0;i<16;i++)
            {
                if (lsn_ncb[num].local_name[i]==ASCII_SP)
                lsn_ncb[num].local_name[i]=0;
```

```
            }
            printf ("\n\t<%s> LISTEN COMMAND....PENDING \n", lsn_ncb[num].local_name);
        }
    }
prt3:   printf ("\n\tHit < Esc > key to CANCEL the pending LISTEN commands.");
    while (!kbhit())  /* loop, if <Esc> key is not hit */
    {
        for (num=0; num<16; num++)
        {
            if (lsn_ncb[num].local_name[0]!=0)
            listen_check (num);
        }
    }
    if (getch ()!=ESC)
        goto prt3;
    for(num=0;num<16;num++)
    {
        if (lsn_ncb[num].local_name[0]!=0)
        clear_pending_cmd(&lsn_ncb[num],num);
    }
    printf ("\n\tHit any key to exit to privious menu");
    while (!kbhit( ));
    getch( );

}

/*  check if any pending LISTEN command is called */
/*  from LAN (near side) or from ISDN (far side).   */

void listen_check (short int l_num)
{
    short int          i;

    switch (lsn_ncb[l_num].cmd_cplt & 0x00ff)
    {
        case CMD_PENDING:  break;
        case GOOD_RETURN:
        {
            if (rec_ncb[l_num].lsn==lsn_ncb[l_num].lsn)
            {
                rec_check (l_num);
            }
            else
            {
                for (i=0;i<16;i++)
                {
                    call_ncb.local_name[i]=lsn_ncb[l_num].remote_name[i];
                    call_ncb.remote_name[i]=lsn_ncb[l_num].local_name[i];
                }
                call_ncb.rto_sto=0x6060; /* time out value */
                for (i=0;i<16;i++)
                {
                    if (call_ncb.local_name[i]==0)
```

```
                call_ncb.local_name[i]=ASCII_SP;
                if (call_ncb.remote_name[i]==0)
                call_ncb.remote_name[i]=ASCII_SP;
            }
            if (lsn_ncb[l_num].lana_num==0)
                call_ncb.lana_num = 1;
            else    call_ncb.lana_num = 0;
            INT_5C (&call_ncb);              /*issue NetBios CALL command */
            rec_ncb[l_num].lsn = lsn_ncb[l_num].lsn;
            rec_ncb[l_num].lana_num = lsn_ncb[l_num].lana_num;
            rec_ncb[l_num].command=REC;
            switch(l_num&0x0001)
            {
                case 0x00: rec_ncb[l_num].buffer=&buff_2[0]; break;
                case 0x01: rec_ncb[l_num].buffer=&buff_1[0]; break;
                /* buffers for two sessions at the same time */
            }
            rec_ncb[l_num].length=0xffff;
            printf("\n\t\tLISTEN ---------->>    GOOD RETURN");
            call_check (l_num);
            INT_5C (&rec_ncb[l_num]);        /* issue RECEIVE command */
        }
    break;
    }
    default:  err_process(& lsn_ncb[l_num],l_num); break;
  }
}

/*  check if a RECEIVE command has been        */
/*  issued or done after a session is established    */

void rec_check (short int r_num)
{
    switch (rec_ncb[r_num].cmd_cplt & 0x00ff)
    {   case CMD_PENDING:
        send_ncb[r_num].length=0;
            break;
        case GOOD_RETURN:

            if (send_ncb[r_num].length!=0 )
            {
            send_check (r_num);
            }
            else
            {
                send_ncb[r_num].command=SEND;
                send_ncb[r_num].buffer = rec_ncb[r_num].buffer;
                send_ncb[r_num].length = rec_ncb[r_num].length;

                if (rec_ncb[r_num].lana_num==0)
                    send_ncb[r_num].lana_num=1;
                else send_ncb[r_num].lana_num=0;
                INT_5C (&send_ncb[r_num]);       /* issue SEND command */
```

```
            }
            break;
        default:  err_process (& rec_ncb[r_num],r_num); break;
    }
}

void call_check (short int c_num)                    /* wait for CALL finishing. */
{
    printf ("\n\t\tCALL ---------->>     PENDING");
    printf("\n\tHit any key to exit");
    while ((call_ncb.cmd_cplt & 0x00ff)== CMD_PENDING)
    {
        if(kbhit()!=0)      { getch();              break;}
    }
    switch (call_ncb.cmd_cplt & 0x00ff)
    {
        case GOOD_RETURN:
        {     printf ("\n\t\tCALL ---------->>     GOOD RETURN\n");
            send_ncb[c_num].lsn = call_ncb.lsn;
            break;
        }
        default: err_process (&call_ncb,c_num); break;
    }
}

void send_check (short int s_num)                    /* Are SEND commands done? */
{
    switch (send_ncb[s_num].cmd_cplt & 0x00ff)
    {
        case CMD_PENDING:                            break;
        case GOOD_RETURN:
        {
            printf("\n\t\tSEND ---------->>     GOOD RETURN");
            send_ncb[s_num].length = 0;
            rec_ncb[s_num].lsn=lsn_ncb[s_num].lsn;
            rec_ncb[s_num].lana_num=lsn_ncb[s_num].lana_num;
            rec_ncb[s_num].length=0xffff;
            INT_5C (&rec_ncb[s_num]);
            break;
        }
        default:
        {
            send_ncb[s_num].length =0;
            err_process (&send_ncb[s_num],s_num);
            break;
        }
    }
}
```

A.4 GW_ERR.C

```
/*****************************************/
/*                                       */
/*    GW_ERR.C -- processing errors.     */
/*     The routine checks the return code */
/*     of a command. If errors exist, the */
/*     extablished session will be closed.*/
/*                                       */
/*    -- It is a part of GATEWAY.EXE     */
/*                                       */
/*****************************************/


#include <conio.h>
#include <stdio.h>
#include "netbios.h"

struct      ncb send_ncb[], lsn_ncb[],   rec_ncb[],
            hang_up_ncb,    cancel_ncb,  call_ncb;

/*********************************/
/*CLEAR PENDING COMMAND() &      */
/*ERROR PROCESS()                */
/*********************************/


void clear_pending_cmd(struct ncb *ptr,short int num )
{
      hang_up_ncb.lsn=send_ncb[num].lsn;
      hang_up_ncb.lana_num=send_ncb[num].lana_num;
      INT_5C(&hang_up_ncb);
      cancel_ncb.buffer=ptr;
      cancel_ncb.lana_num=(*ptr).lana_num;
      cancel_ncb.retcode=0xff;
      INT_5C(&cancel_ncb);
      while (cancel_ncb.retcode&0xff==0xff);
      printf ("\n\t\tCOMMAND CANCEL---->>");

      switch(cancel_ncb.retcode&0xff)
      {
          case 0x00:  printf ("   GOOD RETURN       "); break;
          case 0x03:  printf ("   INVALID_CMD       "); break;
          case 0x21:  printf ("   INTFACE_BUSY      "); break;
          case 0x22:  printf ("   TOO_MANY_CMD      "); break;
          case 0x23:  printf ("   INV_LANA_NUM      "); break;
          case 0x24:  printf ("   CMD_CPLT_W_CNL    "); break;
          case 0x26:  printf ("   CMD_INV_CNL       "); break;
          case 0x31:  printf ("   INTL_ERR          "); break;
          case 0x32:  printf ("   T_NOT_INSTL       "); break;
          case 0x40:
          case 0x41:
```

```
        case 0x42:
        case 0x43:
        case 0x44:
        case 0x45:
        case 0x46:
        case 0x47:
        case 0x48:
        case 0x49:  printf ("   UNUSUAL_NET_CONDITION "); break;
        default:    printf ("   MALFUNCTION           "); break;
    }
}


void err_process(struct ncb *ptr, short int num)
{
    printf ("\n\tnumber: %d / local session number: %d", num,(*ptr).lsn);
    switch ((*ptr).command & 0x00ff)
  {
      case 0xb3: printf ("\n\t SESSION ADAPTER->>   ");      break;
      case 0xb1: printf ("\n\t DELETE NAME------->>   ");    break;
      case 0xb0: printf ("\n\t ADD NAME---------->>   ");    break;
      case 0x97: printf ("\n\t SEND DOUBLE------->>   ");    break;
      case 0x96: printf ("\n\t RECEIVE ANY------->>   ");    break;
      case 0x95: printf ("\n\t RECEIVE---------->>   ");     break;
      case 0x94: printf ("\n\t SEND----------->>   ");       break;
      case 0x91: printf ("\n\t LISTEN---------->>   ");      break;
      case 0x90: printf ("\n\t CALL----------->>   ");       break;
      case 0x35: printf ("\n\t CANCEL--------->>   ");       break;
      case 0x92: printf ("\n\t HANG UP--------->>   ");      break;
      default:   printf ("\n\t ?? command------->>   ");     break;
  }
  printf (" ERROR:");

  switch ((*ptr).cmd_cplt&0x00ff)
  {
      case 0x00:  printf ("   GOOD RETURN           "); break;
      case 0x01:  printf ("   ILEG_BUF_LENGTH       "); break;
      case 0x03:  printf ("     INVALID_CMD         "); break;
      case 0x05:  printf ("     CMD_TIMEOUT         "); break;
      case 0x06:  printf ("     MSG_INCPLT          "); break;
      case 0x08:  printf ("     ILEG_LSN            "); break;
      case 0x09:  printf ("     NO_RESOURCE         "); break;
      case 0x0a:  printf ("   SESS_CLOSE            "); break;
      case 0x0b:  printf ("     CMD_CNL             "); break;
      case 0x0d:  printf ("     DUP_NAME            "); break;
      case 0x0e:  printf ("   NAME_TAB_FULL         "); break;
      case 0x0f:  printf ("     NAME_DEREGTR        "); break;
      case 0x11:  printf ("     L_SESS_TAB_FULL     "); break;
      case 0x12:  printf ("   SESS_OP_REJ           "); break;
      case 0x13:  printf ("     ILEG_NAME_NUM       "); break;
      case 0x14:  printf ("     NO_CALL_NAME        "); break;
      case 0x15:  printf ("   NAME_NOT_FND          "); break;
```

```
    case 0x16:  printf ("    NAME_USED          "); break;
    case 0x17:  printf ("  NAME_DEL             "); break;
    case 0x18:  printf ("  SESS_END_ABNOM       "); break;
    case 0x19:  printf ("    NAME_CONFICT       "); break;
    case 0x1a:  printf ("    INCPLT_REMOTE      "); break;
    case 0x21:  printf ("  INTFACE_BUSY         "); break;
    case 0x22:  printf ("    TOO_MANY_CMD       "); break;
    case 0x23:  printf ("    INV_LANA_NUM       "); break;
    case 0x24:  printf ("  CMD_CPLT_W_CNL       "); break;
    case 0x26:  printf ("    CMD_INV_CNL        "); break;
    case 0x31:  printf ("    INTL_ERR           "); break;
    case 0x32:  printf ("  T_NOT_INSTL          "); break;
    case 0xff:  printf ("    CMD_PENDING        "); break;
    default:    printf ("  MALFUNCTION          "); break;
}

switch ((*ptr).cmd_cplt& 0x00ff)
{
    case GOOD_RETURN:
/*  printf("\t GOOD RETURN !\n");                    */
    break;
    default:
    switch ((*ptr).command & 0x00ff)
    {
        short int i;
        case REC:           /* RECEIVE command */
        case REC_ANY:
            send_ncb[num].length=0;
            hang_up_ncb.lsn=(*ptr).lsn;
            hang_up_ncb.lana_num=(*ptr).lana_num;
            INT_5C (&hang_up_ncb);
                                        /* issue HANG UP command */
            lsn_ncb[num].command=LISTEN;
            lsn_ncb[num].remote_name[0]='*';
            for (i=1;i<16;i++)
            {
                lsn_ncb[num].remote_name[i]=ASCII_SP;
                if (lsn_ncb[num].local_name[i]==0)
                lsn_ncb[num].local_name[i]=ASCII_SP;
            }
            while(hang_up_ncb.cmd_cplt &0x00ff==CMD_PENDING)
            {
                if(kbhit()!=0)
                {
                    getch(); break;
                }
            }
            INT_5C(&lsn_ncb[num]);
            printf ("\n\t<%s> LISTEN COMMAND .....PENDING AGAIN.",
                    lsn_ncb[num].local_name);
            for(i=0;i<16;i++)
            {
                if (lsn_ncb[num].local_name[i]==ASCII_SP)
```

```
            lsn_ncb[num].local_name[i]=0;
        }
        hang_up_ncb.lsn=send_ncb[num].lsn;
        hang_up_ncb.lana_num=send_ncb[num].lana_num;
        INT_5C(&hang_up_ncb);
        while(hang_up_ncb.cmd_cplt &0x00ff==CMD_PENDING)
        {
            if(kbhit()!=0)
            {
                getch(); break;
            }
        }
        break;
    case SEND:                                          /* SEND */
        send_ncb[num].length=0;
        hang_up_ncb.lsn=(*ptr).lsn;
        hang_up_ncb.lana_num=(*ptr).lana_num;
        INT_5C (&hang_up_ncb);
                                                        /* issue HANG UP command  */
        while(hang_up_ncb.cmd_cplt &0x00ff==CMD_PENDING)
        {
            if(kbhit()!=0)
            {
                getch(); break;
            }
        }
        hang_up_ncb.lsn=lsn_ncb[num].lsn;
        hang_up_ncb.lana_num=lsn_ncb[num].lana_num;
        INT_5C (&hang_up_ncb);
                                                        /* issue HANG UP command  */
        lsn_ncb[num].command=0x91;
        lsn_ncb[num].remote_name[0]='*';
        for (i=1;i<16;i++)
        {
            lsn_ncb[num].remote_name[i]=ASCII_SP;
            if (lsn_ncb[num].local_name[i]==0)
            lsn_ncb[num].local_name[i]=ASCII_SP;
        }
        while(hang_up_ncb.cmd_cplt &0x00ff==CMD_PENDING)
        {
            if(kbhit()!=0)
            {
                getch(); break;
            }
        }
        INT_5C(&lsn_ncb[num]);
        printf ("\n\t<%s> LISTEN COMMAND .....PENDING AGAIN.",
                lsn_ncb[num].local_name);
        for(i=0;i<16;i++)
        {
            if (lsn_ncb[num].local_name[i]==ASCII_SP)
            lsn_ncb[num].local_name[i]=0;
        }
```

```
    break;
case LISTEN:            /* LISTEN command */
    hang_up_ncb.lsn=(*ptr).lsn;
    hang_up_ncb.lana_num=(*ptr).lana_num;
    INT_5C (&hang_up_ncb);
                                        /* issue HANG UP command  */
    lsn_ncb[num].command=0x91;
    lsn_ncb[num].remote_name[0]='*';
    for (i=1;i<16;i++)
    {
        lsn_ncb[num].remote_name[i]=ASCII_SP;
        if (lsn_ncb[num].local_name[i]==0)
        lsn_ncb[num].local_name[i]=ASCII_SP;
    }
    while(hang_up_ncb.cmd_cplt &0x00ff==CMD_PENDING)
    {
        if(kbhit()!=0)
        {
        getch(); break;
        }
    }
    INT_5C(&lsn_ncb[num]);
    printf ("\n\t<%s> LISTEN COMMAND .....PENDING AGAIN.",
            lsn_ncb[num].local_name);
    for(i=0;i<16;i++)
    {
        if (lsn_ncb[num].local_name[i]==ASCII_SP)
        lsn_ncb[num].local_name[i]=0;
    }
    break;
 case CALL:
    hang_up_ncb.lsn=lsn_ncb[num].lsn;
    hang_up_ncb.lana_num=lsn_ncb[num].lana_num;
    INT_5C (&hang_up_ncb);
                                        /* issue HANG UP command  */
    lsn_ncb[num].command=0x91;
    lsn_ncb[num].remote_name[0]='*';
    for (i=1;i<16;i++)
    {
        lsn_ncb[num].remote_name[i]=ASCII_SP;
        if (lsn_ncb[num].local_name[i]==0)
        lsn_ncb[num].local_name[i]=ASCII_SP;
    }
    while(hang_up_ncb.cmd_cplt &0x00ff==CMD_PENDING)
    {
        if(kbhit()!=0)
        {
            getch(); break;
        }
    }
    INT_5C(&lsn_ncb[num]);
    printf ("\n\t<%s> LISTEN COMMAND .....PENDING AGAIN.",
            lsn_ncb[num].local_name);
```

```
                for(i=0;i<16;i++)
                {
                    if (lsn_ncb[num].local_name[i]==ASCII_SP)
                    lsn_ncb[num].local_name[i]=0;
                }
                break;
            default:  printf ("\n\t?? command------>>");  break;
        }
        break;
    }
}
```

A.5 NETBIOS.H

```
/*****************************/
/*                           */
/*      NETBIOS.H            */
/*                           */
/*****************************/

#include<stdio.h>

/* definition of the return values*/
#define     GOOD_RETURN          0x00;
#define     ILEG_BUF_LENGTH      0x01;
#define     INVALID_CMD          0x03;
#define     CMD_TIMEOUT          0x05;
#define     MSG_INCPLT           0x06;
#define     ILEG_LSN             0x08;
#define     NO_RESOURCE          0x09;
#define     SESS_CLOSE           0x0a;
#define     CMD_CNL              0x0b;
#define     DUP_NAME             0x0d;
#define     NAME_TAB_FULL        0x0e;
#define     NAME_DEREGTR         0x0f;
#define     L_SESS_TAB_FULL      0x11;
#define     SESS_OP_REJ          0x12;
#define     ILEG_NAME_NUM        0x13;
#define     NO_CALL_NAME         0x14;
#define     NAME_NOT_FND         0x15;
#define     NAME_USED            0x16;
#define     NAME_DEL             0x17;
#define     SESS_END_ABNOM       0x18;
#define     NAME_CONFICT         0x19;
#define     INCPLT_REMOTE        0x1a;
#define     INTFACE_BUSY         0x21;
#define     TOO_MANY_CMD         0x22;
#define     INV_LANA_NUM         0x23;
#define     CMD_CPLT_W_CNL       0x24;
#define     CMD_INV_CNL          0x26;
#define     INTL_ERR             0x31;
#define     T_NOT_INSTL          0x32;
#define     CMD_PENDING          0xff;


/* definition of the session commands */
#define     CALL          0x90;
#define     LISTEN        0x91;
#define     HANGUP        0x92;
#define     SEND          0x94;
#define     REC           0x95;
#define     REC_ANY       0x96;
#define     SEND_DOB      0x97;
#define     ADD_NAME      0xb0;
```

```
#define     DEL_NAME      0xb1;
#define     SESS_ADPT     0xb3;
#define     CANCEL        0x35;


/*  definition of constants*/
#define     ZERO                        0;
#define     ASCII_ZERO                  0x30;
#define     ASCII_SP                    0x20;
#define     NON_INIT                    0x00;
#define     INIT                        0x01;
#define     ESC                         0x1b;
#define     ERROR                       0;
#define     OK                          1;

#define     INITIAL                     '1';
#define     RUN_GW                      '2';
#define     ADD_DEL_LAN_NAME_A          'A';
#define     ADD_DEL_LAN_NAME_AA         'a';
#define     ADD_DEL_SESS_NAME_B         'B';
#define     ADD_DEL_SESS_NAME_BB        'b';
#define     MOD_NAME_R                  'R';
#define     MOD_NAME_RR                 'r';
#define     CHK_NAME_C                  'C';
#define     CHK_NAME_CC                 'c';
#define     REMOTE_NAME_R               'R';
#define     REMOTE_NAME_RR              'r';
#define     LOCAL_NAME_L                'L';
#define     LOCAL_NAME_LL               'l';


struct ncb {
            char        command;
            char        retcode;
            char        lsn;
            char        num;
            long int    buffer;
            short int   length;
            char        remote_name[16];
            char        local_name[16];
            short int   rto_sto;
            long int    post;
            char        lana_num;
            char        cmd_cplt;
            long int    reserve_1;          /* 4 bytes */
            long int    reserve_2;
            long int    reserve_3;
            short int   reserved_4;
            }
```

```
/*********************************/
/* Initialization of NCB feilds  */
/*********************************/
struct ncb lsn_lan_ncb[4] =          {0};
struct ncb lsn_isdn_ncb[4] =         {0};
struct ncb del_name_ncb =            {0xb1,0};
struct ncb add_name_ncb =            {0xb0,0};
struct ncb send_double_ncb =         {0x97,0};
struct ncb rec_any_ncb =             {0x96,0};
struct ncb rec_ncb[16] =             {0x95,0};
struct ncb send_ncb[16] =            {0x94,0};
struct ncb hang_up_ncb =             {0x92,0};
struct ncb lsn_ncb[16] =             {0x91,0};
struct ncb call_ncb =                {0x90,0};
struct ncb cancel_ncb =              {0x35,0};
```

A.6 INT_5C.ASM

```
;
; INT_5C.ASM  --executing NetBIOS commands.
;           --compiled with MSAM 4.0.



DGROUP  GROUP   _DATA
        ASSUME  es: DGROUP
_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS
_TEXT   SEGMENT WORD PUBLIC 'CODE'
        ASSUME cs: _TEXT
        PUBLIC _INT_5C
_INT_5C PROC    NEAR
        push    ds
        pop     es
        push    bp
        mov     bp,     sp

        mov     bx,     [bp+4]
        int     5ch

        pop     bp
        ret
_INT_5C ENDP
_TEXT   ENDS
        END
```

Appendix B

Testing Program

B.1 BM.C

```
/*************************************************/
/*  BM.C--A benchmark program for the test of end-to-end  */
/*          data transfer.                                */
/*       --BM.EXE contains BM.C and INT_5C.ASM            */
/*       --Compiled with MS C  v.5.0.                     */
/*************************************************/

#include <stdlib.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <time.h>
#include <conio.h>
#include "netbios.h"

/* the order of all routines in this file */
void main (void);
void menu (short int lan_num);
void lsn (viod);                    /* process a listerner */
void call (void);                   /* process a caller */
void rec(int x);                    /* receive data on an established sesssion */
void r_dg( void);                   /* receive datagram */
void send (int y)   ;               /* send data on an established session */
viod s_dg ( void);                                  /* send datagram */
void receiving (rf,unsigned int bloc);              /* For disk-to-disk data receiving */
void receiving_1 (void) ;                           /* For memory-to-memory data receiving */
viod rec_datagram (rf);
void select_s_msg_size(unsigned int *bloc_ptr, unsigned long *l_block_ptr);
void select_d_msg_size(unsigned int *bloc_ptr, unsigned long *l_block_ptr);
void sending (sf, unsigned int block);              /* For disk-to-disk data sending */
void sending_1 (unsigned int block);                /* For memory-to-memory data sending */

/* all issued routines */

void send_datagram (unsigned int block);
void err_process(struct ncb *ptr);

struct timeb    tbufs1, tbufs2, tbufr1, tbufr2;

unsigned long       rcnt,   scnt;
char                rrcnt[20];
```

```
char            sscnt[20];
char            buff [0xffff]; /*buffer size 64K-1 bytes */
char            *rp,    *sp;
int             radix=10;

void main (void)
{
    char    input=ZERO;
    short int    in_lana;
    while ((input & 0x00fe) != ASCII_ZERO)
    {
        printf ("\n\tSelect the adapter number < 0 > or < 1 >:  ");
        scanf ("%c",&input);
    }
    in_lana=input&0x0001;
    add_name_ncb.lana_num=in_lana;
    del_name_ncb.lana_num=in_lana;
    lsn_ncb.lana_num=in_lana;
    rec_ncb.lana_num=in_lana;
    send_ncb.lana_num=in_lana;
    rec_dg_ncb.lana_num=in_lana;
    send_dg_ncb.lana_num=in_lana;
    call_ncb.lana_num=in_lana;
    hang_up_ncb.lana_num=in_lana;
    menu (in_lana);
}

void menu (short int lan_num)
{
    char        in_char;
    while (in_char!=ESC)
    {
        printf ("\n\n\n\t\tLAN adapter number is %d",lan_num);
        printf ("\n\n\t\tNetBIOS commands for test -- MENU_1");
        printf("\nName Support:");
        printf ("\n\t\t< 1 >--ADD NAME        < 2 >--DELETE NAME");
        printf("\nSession Support:");
        printf ("\n\t\t< 3 >--LISTEN          < 4 >--CALL");
        printf ("\n\t\t    ( with data receiving and sending )");
        printf("\nDatagram Support:");
        printf ("\n\t\t< 5 >--RECEIVE_DG      < 6 >--SEND_DG");
        printf ("\nGeneral support:");
        printf ("\n\t\t< R >--RESET          <Esc> ===> Exit to DOS");
        printf ("\n\n\tYOUR  SELECTION  ==============>>  ");
        in_char=getche();
        switch (in_char)
        {
            case '1':   add_name_issued( );      break; /* add a name */
            case '2':   del_name_issued( );      break; /* delete a name */
            case '3':   lsn ( );                 break; /* session listen */
            case '4':   call ( );                break; /* session call */
            case '5':   r_dg ( );                break; /* receive datagram */
```

```
            case '6':    s_dg ( );                  break; /* send gatagram */
            case 'r':
            case 'R':    reset_issued ( );          break; /* reset the adapter */
            case ESC:                               break; /* exit from MAIN program to DOS. */
            default:                                break; /* return to the menu above */
        }
    }
    exit(0); /* exit to DOS */
}


void lsn (viod)                                     /* process a listerner */
{
    if (lsn_issued ()!=OK)
    printf("\n\Session establishment is incorrect!");
    else
    {
prt1:   printf("\n\nSession has been established,the session # is =======>%d", lsn_ncb.lsn);
        printf("\n\n\tJOB SELECTIONS      < R > =======> RECEIVE DATA;");
        printf("\n\n\t                < S > =======> SEND DATA;");
        printf("\n\n\t< Esc > =======> CLOSE this session and exit to main MENU_1.\n");
        printf("\t========>");
        switch(getche())
        {
            case'R':
            case'r':    rec (1); goto prt1; /* receive data on the session */
            case'S':
            case's':    send (1); goto prt1; /* send data on the session */
            case ESC:
            {
                hang_up_ncb.lsn=lsn_ncb.lsn; /* close the established session */
                hang_up_issued ();
                break;
            }
            default: goto prt1; /* display the above menu */
        }
    }
}


void call (void)                                    /* process a caller */
{
    if (call_issued ()!=OK)
    printf("\n\Session establishment is incorrect!");
    else
    {
prt2:   printf("\n\nSession has been established,the session # is =======> %d", call_ncb.lsn);
        printf("\n\n\tJOB SELECTIONS      < R > =======> RECEIVE DATA;");
        printf("\n\n\t                < S > =======> SEND DATA;");
        printf("\n\n\t< Esc > =======> CLOSE this session and exit to main MENU_1.\n");
```

```
        printf("\t=========>");
        switch(getche())
        {
            case'R':
            case'r':    rec (2);   goto prt2; /* receive data on the session */
            case'S':
            case's':    send (2); goto prt2; /* send data on the session */
            case ESC:
            {
                hang_up_ncb.lsn=call_ncb.lsn; /* close the established session */
                hang_up_issued ();
                break;
            }
            default: goto prt2; /* display the above menu again */
        }
    }
}


void rec(int x)                    /* receive data on an established sesssion */
/* x = 1: receiver is a listener */
/* x = 2: receiver is a caller */
{
    short int    i, m, rflag;
    unsigned int bloc;
    unsigned long l_block;
    FILE   *rf;

    rcnt=ZERO;
    select_s_msg_size(&bloc, &l_block); /* select the size of a data block */
    rf = fopen ("rec.dat","wb"); /* read a file */
    if (rf==NULL)
    printf("\n\trec.dat open ERROR\n"); /* file does exist */
    else /* file exists */
    {
        if(x==1) /* the listener is receiving */
        rec_ncb.lsn=lsn_ncb.lsn;
        else /* the caller is receiving */
        rec_ncb.lsn=call_ncb.lsn;
        ftime(&tbufr1); /* counting the start time */
        for (i=0;i<n;i++)
        {
            rflag=receiving_1 ( );              /* memory-to-memory */
/*          rflag=receiving (rf.bloc);  */      /* disk-to-disk */
            if (rflag==ERROR) break;
        }
        ftime(&tbufr2); /* counting the stop time */
        printf("\nStarting to receive at the time of %.19s(s) & %hu(ms)\n",
                ctime(&(tbufr1.time)),tbufr1.millitm);
        printf("\Ending to receive at the time of %.19s(s) & %hu(ms)\n",
                ctime(&(tbufr2.time)),tbufr2.millitm);
        printf("Receiving data takes %.f(s) & %.f (ms)\n", difftime(tbufr2.time,tbufr1.time),
```

```
                difftime(tbufr2.millitm,tbufr1.millitm));
        fclose (rf); /close the file */
        rcnt=l_block;
        rcnt*=i;
        printf ("\n\tFile has been received ==> ");
        if (rflag==OK)
        {
            printf("OK!\n");
            rp=ltoa(rcnt,rrcnt,radix); /* convert the long intergal to characters */
            printf("\tTotal length of receiving data is %s bytes",rrcnt);
        }
        else
        printf ("with ERROR!\n");
    }
    printf("\n\tPress any key to select again");
    getche();
}

void r_dg( void)                                    /* receive datagram */
{
    short int    i, m, rflag;
    unsigned int bloc;
    unsigned long l_block;
    char    c;
    FILE   *rf;
    rcnt=ZERO;
    select_d_msg_size(&bloc, &l_block); select the size of a datagram message size */
    rf = fopen ("rec.dat","wb");
    if (rf==NULL)
    printf("\n\trec.dat open error\n");
    else
    {
            /* get start time */
        ftime(&tbufr1);
        for (i=0;i<n;i++)
        {
            rflag=rec_datagram (rf);
            if (rflag==ERROR) break;
        }
            /* get stop time */
        ftime(&tbufr2);
            /* show time */
        printf("\nStarting to receive datagram at the time of %.19s(s) & %hu(ms)\n",
                ctime(&(tbufr1.time)),tbufr1.millitm);
        printf("\Ending to receive datagram at the time of %.19s(s) & %hu(ms)\n",
                ctime(&(tbufr2.time)),tbufr2.millitm);
        printf("Receiving datagram takes %.f(s) & %.f (ms)\n", difftime(tbufr2.time,tbufr1.time),
                difftime(tbufr2.millitm,tbufr1.millitm));
        fclose (rf);
        rcnt=l_block;
        rcnt*=i;
        printf ("\n\tFile has been received ==> ");
```

```
        if (rflag==OK)
        {
            printf("OK!\n");
            rp=ltoa(rcnt,rrcnt,radix);
            printf("\tTotal length of receiving data is %s bytes",rrcnt);
        }
        else
        printf ("with ERROR!\n");
    }
    printf("\n\tPress any key to select again");
    getche();
}

void send (int y)                   /* send data on an established session */
{
    short int    i,sflag;
    unsigned int bloc;
    float    d_block;
    unsigned long l_block;
    FILE    *sf;
    scnt=ZERO;
    select_s_msg_size(&bloc, &l_block); /* select the size of a data block */
    sf = fopen ("send.dat","rb"); /* open a data file */
    if (sf==NULL)
    printf("\n\tsend.dat open error\n");
    else
    {   if(y==1) /* sender is a listener */
        send_ncb.lsn=lsn_ncb.lsn;
        else /* sender is a caller */
        send_ncb.lsn=call_ncb.lsn;
        ftime(&tbufs1); /* start sending a file */
        for (i=0;i<n;i++)
        {
            sflag=sending_1 (bloc);             /* memory-to-memory */
/*          sflag=sending (sf, bloc);           disk-to-disk */
            if (sflag!=OK) break;
        }
        ftime(&tbufs2); /* stop sending a file */
        printf("\nStarting to send at the time of %.19s(s) & %hu(ms)\n", ctime(&(tbufs1.time)),
                tbufs1.millitm);
        printf("\Ending to send at the time of %.19s(s) & %hu(ms)\n",
                ctime(&(tbufs2.time)),tbufs2.millitm);
        printf("Sending data takes %.f(s) & %.f(ms)\n", difftime(tbufs2.time,tbufs1.time),
                difftime(tbufs2.millitm, tbufs1.millitm));
        scnt=l_block;
        scnt*=i;
        fclose (sf);
        printf ("\n\tFile has been sent ==> ");
        if (sflag==OK || sflag==EOF)
        {   printf("OK!");
            sp=ltoa(scnt,sscnt,radix);
            printf ("\n\tTotal length of sending data is %s bytes",sscnt);
```

```
        }
        else
        {
            printf("with ERROR!");
        }

    }
    printf("\n\tPress any key to select again");
    getche();
}

viod s_dg ( void)                                   /* send datagram */
{
        short int    i, sflag;
        unsigned int bloc;
        unsigned long l_block;
        FILE   *sf;

        scnt=ZERO;
        for(i=0;i<16;i++)
        {
            if (send_dg_ncb.remote_name[i]==ASCII_SP )
            send_dg_ncb.remote_name[i]=ZERO;
        }
                /* enter the name of a remote station */
        printf ("\n\n\tThe old remote name------>> <%s>",send_dg_ncb.remote_name);
        printf ("\n\tEnter new remote NAME -->> ");
        scanf ("%s",send_dg_ncb.remote_name);
        for(i=0;i<15;i++)
        {
            if (send_dg_ncb.remote_name[i]==ZERO)
            {   send_dg_ncb.remote_name[i]=ASCII_SP;
                send_dg_ncb.remote_name[i+1]=ZERO;
            }
        }
        send_dg_ncb.remote_name[15]=ASCII_SP;

            /* select the size of a datagram data block */
        select_d_msg_size(&bloc,&l_block);
        sf = fopen ("send.dat","rb");
        if (sf==NULL)
        printf("\n\tsend.dat open error\n");
            else
        {
            ftime(&tbufs1);
            for (i=0;i<n;i++)
            {
                sflag=send_datagram (bloc);
                if (sflag!=OK) break;
            }
            ftime(&tbufs2);
            printf("\nStarting to send datagram at the time of %.19s(s) & %hu(ms)\n",
```

```
                ctime(&(tbufs1.time)), tbufs1.millitm);
            printf("\Ending to send datagram at the time of %.19s(s) & %hu(ms)\n",
                ctime(&(tbufs2.time)), tbufs2.millitm);
            printf("Sending datagram takes %.f(s) & %.f(ms)\n", difftime(tbufs2.time,tbufs1.
            time), difftime(tbufs2.millitm,tbufs1.millitm));
            scnt=l_block;
            scnt*=i;
            fclose (sf);
            printf ("\n\tFile has been sent ==> ");
            if (sflag==OK || sflag==EOF)
            {   printf("OK!");
                sp=ltoa(scnt,sscnt,radix);
                printf ("\n\tTotal length of sending data is %s bytes",sscnt);
            }
            else            printf("with ERROR!");
        }
        printf("\n\tPress any key to select again");
        getche();
}

void receiving (rf,unsigned int bloc)           /* For disk-to-disk data receiving */
FILE   *rf;
{
    unsigned int   k;
    short int      inchar;
/*  printf ("\nlsn=%d",rec_ncb.lsn);*/
    if(rec_issued ()!=OK) return ERROR;
    else
    {/* printf("\nrec length=%x",rec_ncb.length);*/
        for (k=0;k<rec_ncb.length;k++)
        {   inchar=buff[k]&0x00ff;
            rcnt++;
        putc(inchar,rf);
        }
        if (rec_ncb.length<bloc) return EOF;
        return OK;
    }
}

void receiving_1 (void)                         /* For memory-to-memory data receiving */
{
    short int    k;
    short int      inchar;
    if(rec_issued ()!=OK) return ERROR;
    return OK;
}

viod rec_datagram (rf)
FILE   *rf;
{
    if(rec_dg_issued ()!=OK) return ERROR;
    return OK;
```

```
}

    /* select the size of a datagram data block*/

void select_d_msg_size(unsigned int *bloc_ptr, unsigned long *l_block_ptr)
unsigned int *bloc_ptr;
unsigned long *l_block_ptr;
{
    char    c;
    short int  n, in_block;

        printf("\n");
sel_b:  printf("\n\t\tSELECT THE SIZE OF A DATA BLOCK");
        printf("\n\tA data block = N x 1 bytes,  N (1 to 512) is-->", min_size);
        scanf ("%d", &in_block);
        if(in_block<1 || in_block>512)
        {
            printf("\n\tThis is an illegal number, try again!");
            goto sel_b;
        }
        *bloc_ptr=in_block;
        *l_block_ptr = in_block;

sel_p:  printf("\n\tThe amount of data sent are n*%u k bytes, n (1--65535) is-->",in_block);
        if (scanf ("%d",&n)==ZERO)
        {
            scanf ("%s",&c);
            printf ("\n\t This is an illegal number, try again!");
            goto sel_p;
        }
        else if (n < 1 || n > 65535)
        {
            printf ("\n\t The < n > is too small or too large!");
            goto sel_p;
        }
}


/* select a session Message size */

void select_s_msg_size(unsigned int *bloc_ptr, unsigned long *l_block_ptr)
{
    short int n, in_block;
    float  d_block;
    char    c, add1;

        printf("\n");
sel_1:  printf("\n\t\tSELECT THE SIZE OF A DATA BLOCK");
        printf("\n\tA data block = N x 0.5k bytes, N (1 to 128) is--> ");
        scanf ("%d", &in_block);
        d_block=0.5*in_block;
        if ( in_block<1 || in_block>128)
```

```
        {
            printf("\n\tThis is an illegal number, try again!");
            goto sel_1;
        }
        *bloc_ptr=in_block;
        (*bloc_ptr)*=512;
        *l_block_ptr =in_block;
        (*l_block_ptr)*=512;
        if(in_block==128)
        {
            (*bloc_ptr)--;
            (*l_block_ptr)--;
        }
sel_2:  printf("\n\tThe amount of data received are n*%.1f k bytes, n (1--4000) is-->",d_block);
        if (scanf ("%d",&n)==ZERO)
        {
            scanf ("%s",&c);
            printf ("\n\t This is an illegal number, try again!");
            goto sel_2;
        }
        else if (n < 1 || n > 4000)
        {
            printf ("\n\t The < n > is too small or too large!");
            goto sel_2;
        }
sel_3:  printf("\n\tDo you want to add 1 more byte on the data block?");
        printf("\n\t < Y >----add 1 byte,    < any key >----start");
        add1=getche();
        if (add1=='y'|| add1=='Y')
        {
            if (in_block!=128)
            {
                (*l_block_ptr)++;
                (*bloc_ptr)++;
            }
            printf("\n\tBlock size = %u bytes",*bloc_ptr);
            goto sel_3;
        }
}


void sending (sf, unsigned int block)          /* For disk-to-disk data sending */
FILE   *sf;
    {
        short int    k;
        short int    outchar;
        for (k=0;k<block;k++)
        {   outchar=getc(sf);

            if (outchar==EOF)break;
            buff[k]=outchar;
            scnt++;
```

```
        }
        send_ncb.length=k;
/*      printf("\nlsn=%d\n",send_ncb.lsn);*/
        if (send_issued ()!=OK)  return ERROR;
        else if  (outchar==EOF)  return EOF;
        return OK;


}

void sending_1 (unsigned int block)              /* For memory-to-memory data sending */
{
        send_ncb.length=bloc-1;
        if (send_issued ()!=OK) return ERROR;
        return OK;
}

void send_datagram (unsigned int block)
{
        send_ncb.length=block-1;
        if (send_dg_issued ()!=OK) return ERROR;
        return OK;
}

void add_name_issued (void)
{
    short int i;

    printf ("\n\tThe name you added last time------>>%s",add_name_ncb.local_name);
    printf ("\n\n\tAdd local NAME <%s> in LAN card (Sytek or
                                                    Ethernet)",add_name_ncb.local_name);
    scanf ("%s",add_name_ncb.local_name);
    for(i=0;i<16;i++)
    {
        if (add_name_ncb.local_name[i]==ZERO )
        {   add_name_ncb.local_name[i]=ASCII_SP;
            add_name_ncb.local_name[i+1]=ZERO;
        }
    }
    INT_5C (&add_name_ncb);
        /* issue ADD_NAME command  */
    err_process(&add_name_ncb);
    for(i=0;i<16;i++)
    {
        if (add_name_ncb.local_name[i]==ASCII_SP )
        add_name_ncb.local_name[i]=ZERO;
    }
}


void del_name_issued (void)
{
```

```
    short int i;

    {
        printf ("\n\tThe name you deleted last time------>>%s",del_name_ncb.local_name);
        printf ("\n\tEnter NAME which you want to delete-->> ");
        scanf ("%s",del_name_ncb.local_name);
    }
    printf ("\n\n\t\tDelete local NAME <%s> in LAN card (Sytek or Ethernet)",
            del_name_ncb.local_name);
    for(i=0;i<16;i++)
    {
        if (del_name_ncb.local_name[i]==ZERO)
        {   del_name_ncb.local_name[i]=ASCII_SP;
            del_name_ncb.local_name[i+1]=ZERO;
        }
    }
    INT_5C (&del_name_ncb);
        /* issue DEL_NAME command */
    err_process(&del_name_ncb);
    for(i=0;i<16;i++)
    {
        if (del_name_ncb.local_name[i]==ASCII_SP )
        del_name_ncb.local_name[i]=ZERO;
    }
}



void lsn_issued (void)
{
    short int i;

    printf ("\n\n\tEnter NAME which listens-->> ");
    scanf ("%s",lsn_ncb.local_name);
    for(i=0;i<16;i++)
    {
        lsn_ncb.remote_name[i]=ASCII_SP;
        if (lsn_ncb.local_name[i]==ZERO)
        {   lsn_ncb.local_name[i]=ASCII_SP;
            lsn_ncb.local_name[i+1]=ZERO;
        }
    }
    lsn_ncb.remote_name[0]='*';
    lsn_ncb.rto_sto=0x6060;
    INT_5C (&lsn_ncb);                      /* issue LISTEN command */
    for(i=0;i<16;i++)
    {
        if (lsn_ncb.local_name[i]==ASCII_SP )
        lsn_ncb.local_name[i]=ZERO;
    }
    if (err_process(&lsn_ncb)==OK)          return OK;
    else                                    return ERROR;
```

```
}

void rec_issued(void)
{
    struct timeb timebufr1;
    struct timeb timebufr2;
    rec_ncb.buffer =&buff[0];
    rec_ncb.length =0xffff;
    INT_5C (&rec_ncb);                          /* issue RECEIVE command  */
    if (err_process(&rec_ncb)==OK)
    return OK;
    else     return ERROR;
}

void rec_dg_issued (void)
{
    struct timeb timebufr1;
    struct timeb timebufr2;
  rec_dg_ncb.num =0xff;
    rec_dg_ncb.buffer =&buff[0];
    rec_dg_ncb.length =0x02ff;
    INT_5C (&rec_dg_ncb);                       /* issue RECEIVE DATAGRAM command  */
    if (err_process(&rec_dg_ncb)==OK)           return OK;
    else                                        return ERROR;
}

void call_issued (void)
{
    short int i;

    for(i=0;i<16;i++)
    {
        if (call_ncb.local_name[i]==ASCII_SP )
        call_ncb.local_name[i]=ZERO;
        if (call_ncb.remote_name[i]==ASCII_SP )
        call_ncb.remote_name[i]=ZERO;
    }
        printf ("\n\n\tThe old remote name------>> <%s>",call_ncb.remote_name);
        printf ("\n\tEnter new remote NAME -->> ");
        scanf ("%s",call_ncb.remote_name);
        printf ("\n\tThe old local name------>> <%s>",call_ncb.local_name);
        printf ("\n\tEnter new local NAME -->> ");
        scanf ("%s",call_ncb.local_name);
    for(i=0;i<15;i++)
    {
        if (call_ncb.local_name[i]==ZERO )
        {   call_ncb.local_name[i]=ASCII_SP;
            call_ncb.local_name[i+1]=ZERO;
        }
        if (call_ncb.remote_name[i]==ZERO)
        {   call_ncb.remote_name[i]=ASCII_SP;
            call_ncb.remote_name[i+1]=ZERO;
```

```
        }
    }
    call_ncb.local_name[15]=ASCII_SP;
    call_ncb.remote_name[15]=ASCII_SP;
    call_ncb.rto_sto=0x6060;
    INT_5C (&call_ncb);                    /* issue CALL command */
    if (err_process(&call_ncb)==OK)
    return OK;
    else    return ERROR;
}

void send_issued (void)
{
    struct timeb timebufs1;
    struct timeb timebufs2;
    send_ncb.buffer = &buff[0];
    INT_5C (&send_ncb);
    /* issue SEND command */
    if (err_process(&send_ncb)==OK)
        return OK;
    else    return ERROR;
}

void send_dg_issued (void)
{
    struct timeb timebufs1;
    struct timeb timebufs2;
    send_dg_ncb.num=add_name_ncb.num;
    send_dg_ncb.buffer = &buff[0];
    INT_5C (&send_dg_ncb);
        /* issue SEND DATAGRAM command */
    if (err_process(&send_dg_ncb)==OK)
        return OK;
    else    return ERROR;
}

void hang_up_issued (void)
{
    INT_5C (&hang_up_ncb);
        /* issue HANG UP command */
    if (err_process(&hang_up_ncb)==OK)      return OK;
    else    return ERROR;
}

void reset_issued (void)
{   printf ("\n\tThe defult values are 6 sessions and 12 commands");
    printf ("\n\tNumber of sessions to be supported ( 1 to 32 )-->> ");
    scanf ("%d", &reset_ncb.lsn);
    printf ("\n\tNumber of command blocks to be supported ( 1 to 32 )-->> ");
    scanf ("%d", &reset_ncb.num);
    INT_5C (&reset_ncb);                   /* issue RESET command */
    while (reset_ncb.retcode==0xff)
```

```
    if(kbhit()!=0)  break;
    switch (reset_ncb.retcode)
    {
        case 0x00: printf ("\n\tRESET command OK !"); break;
        case 0x03:  printf ("\n\tinvalid command when RESET"); break;
        case 0x23: printf ("\n\tInvalid lana_num when RESET issued"); break;
        case 0xff:  printf ("\n\tRESET always pending"); break;
        default: printf ("\n\tRESET in unusual network condition or malfunction"); break;
    }
}

void err_process(struct ncb *ptr)
{
/*  printf ("\n\tcommand is %x", (*ptr).command);*/
switch ((*ptr).command & 0x00ff)
    {
        case 0xb3: printf ("\n\tSESSION ADAPTER->>");          break;
        case 0xb1: printf ("\n\tDELETE NAME------->>");        break;
        case  0xb0 :  printf ("\n\tADD NAME-------->>");       break;
        case 0x97: printf ("\n\tSEND DOUBLE------->>");        break;
        case 0x96: printf ("\n\tRECEIVE ANY------->>");        break;
        case 0x95: /* printf ("\n\tRECEIVE---------->>"); */  break;
        case 0x94:/* printf ("\n\tSEND---------->>"); */      break;
        case 0x91: printf ("\n\tLISTEN---------->>");         break;
        case 0x90: printf ("\n\tCALL---------->>");           break;
        case 0x35: printf ("\n\tCANCEL---------->>");         break;
        case 0x92: printf ("\n\tHANG UP---------->>");        break;
        default: /*  printf ("\n\t?? command---------->>"); */  break;
    }
while (((*ptr).cmd_cplt & 0x00ff)==0xff)
{
    if(kbhit()!=0)
    break;
}
switch ((*ptr).cmd_cplt& 0x00ff)
    {
    case    GOOD_RETURN:
    return OK;      break;
    default:
        printf (" ERROR:");    break;
    }
switch ((*ptr).cmd_cplt&0x00ff)
{
   case 0x00:   break;
   case 0x01:   printf ("   ILEG_BUF_LENGTH    "); break;
   case 0x03:   printf ("   INVALID_CMD        "); break;
   case 0x05:   printf ("   CMD_TIMEOUT        "); break;
   case 0x06:   printf ("   MSG_INCPLT         "); break;
   case 0x08:   printf ("   ILEG_LSN           "); break;
   case 0x09:   printf ("   NO_RESOURCE        "); break;
   case 0x0a:   printf ("   SESS_CLOSE         "); break;
   case 0x0b:   printf ("   CMD_CNL            "); break;
```

```
    case 0x0d:   printf ("   DUP_NAME              "); break;
    case 0x0e:   printf ("   NAME_TAB_FULL         "); break;
    case 0x0f:   printf ("   NAME_DEREGTR          "); break;
    case 0x11:   printf ("   L_SESS_TAB_FULL       "); break;
    case 0x12:   printf ("   SESS_OP_REJ           "); break;
    case 0x13:   printf ("   ILEG_NAME_NUM         "); break;
    case 0x14:   printf ("   NO_CALL_NAME          "); break;
    case 0x15:   printf ("   NAME_NOT_FND          "); break;
    case 0x16:   printf ("   NAME_USED             "); break;
    case 0x17:   printf ("   NAME_DEL              "); break;
    case 0x18:   printf ("   SESS_END_ABNOM        "); break;
    case 0x19:   printf ("   NAME_CONFICT          "); break;
    case 0x1a:   printf ("   INCPLT_REMOTE         "); break;
    case 0x21:   printf ("   INTFACE_BUSY          "); break;
    case 0x22:   printf ("   TOO_MANY_CMD          "); break;
    case 0x23:   printf ("   INV_LANA_NUM          "); break;
    case 0x24:   printf ("   CMD_CPLT_W_CNL        "); break;
    case 0x26:   printf ("   CMD_INV_CNL           "); break;
    case 0x31:   printf ("   INTL_ERR              "); break;
    case 0x32:   printf ("      T_NOT_INSTL        "); break;
    case 0xff:   printf ("   CMD_PENDING           ");
         cancel_ncb.lana_num=(*ptr).lana_num;
         cancel_ncb.buffer=ptr;
         cancel_ncb.retcode=0xff;
         INT_5C(&cancel_ncb);
         while (cancel_ncb.retcode&0xff==0xff)
         {       if(kbhit()!=0) getch(); break;
         }
         printf ("\n\tCOMMAND CANCEL-------->");
         switch(cancel_ncb.retcode&0xff)
         {
         case 0x00:   printf ("   GOOD RETURN     "); break;
         case 0x03:   printf ("   INVALID_CMD     "); break;
         case 0x21:   printf ("   INTFACE_BUSY    "); break;
         case 0x22:   printf ("   TOO_MANY_CMD    "); break;
         case 0x23:   printf ("   INV_LANA_NUM    "); break;
         case 0x24:   printf ("   CMD_CPLT_W_CNL  "); break;
         case 0x26:   printf ("   CMD_INV_CNL     "); break;
         case 0x31:   printf ("   INTL_ERR        "); break;
         case 0x32:   printf ("   T_NOT_INSTL     "); break;
         case 0x40: case 0x41: case 0x42: case 0x43: case 0x44:
         case 0x45: case 0x46: case 0x47: case 0x48:
         case 0x49:   printf ("   UNUSUAL_NET_CONDITION "); break;
         case 0xff:   printf ("   PENDING         "); break;
         default:     printf ("      MALFUNCTION  "); break;
         }
         break;
    default:     printf ("   MALFUNCTION          "); break;
    }
    if ((*ptr).cmd_cplt!=0x00)                      return ERROR;
}
```