USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5000

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

Protocol Interoperability Between DDN and ISO Protocols

(ASQBG-C-89-021)

August 1988

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704--188
Exp. Date: Jun 30, 1

- **1a. REPORT SECURITY CLASSIFICATION:** UNCLASSIFIED
- **1b. RESTRICTIVE MARKINGS:** NONE
- **2a. SECURITY CLASSIFICATION AUTHORITY:** N/A
- **2b. DECLASSIFICATION / DOUWNGRADING SCHEDULE:** N/A
- **3. DISTRIBUTION / AVAILABILITY OF REPORT:** UNLIMITED
- **4. PERFORMING ORGANIZATION REPORT NUMBER(S):** N/A
- **5. MONITORING ORGANIZATION REPORT NUMBER(S):** ASQBG-C-89-021
- **6a. NAME OF PERFORMING ORGANIZATION:** University of Arizona
- **6b. OFFICE SYMBOL (if applicable):**
- **7a. NAME OF MONITORING ORGANIZATION:** AIRMICS
- **6c. ADDRESS (City, State, and ZIP Code):** Computer Engineering Research Laboratory, Electrical and Computer Engineering Depart., University of Arizona, Tucson, Arizona 85721
- **7b. ADDRESS (City, State, and Zip Code):** 115 O'Keefe Bldg., Georgia Institute of Technology, Atlanta, GA 30332-0800
- **8a. NAME OF FUNDING/SPONSORING ORGANIZATION:** AIRMICS
- **8b. OFFICE SYMBOL (if applicable):** ASQBG - C
- **9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:**
- **8c. ADDRESS (City, State, and ZIP Code):** 115 O'Keefe Bldg., Georgia Institute of Technology, Atlanta, GA 30332-0800

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
62783A	DY10	00-08	

- **11. TITLE (Include Security Classification):** Protocol Interoperability Between DDN and ISO Protocols (UNCLASSIFIED)
- **12. PERSONAL AUTHOR(S):** JianYi Tao and Ralph Martinez
- **13a. TYPE OF REPORT:**
- **13b. TIME COVERED:** FROM ______ TO ______
- **14. DATE OF REPORT (Year, Month, Day):** 1988, August
- **15. PAGE COUNT:** 276
- **16. SUPPLEMENTARY NOTATION:**

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

DDN Protocols, ISO Protocols, CCITT Recommendations, Transport Gateways, DDN Internet Architecture, ISO/DDN Interoperability

19. ABSTRACT (Continue on reverse if necessary and identity by block number)

This study focuses on the following four problem areas: 1) the general issues involved in protocol conversion, 2) protocol conversion in the DDN to ISO environment, 3) a detailed understanding of both protocol suites, and 4) approaches to achieve interoperability between TCP/IP and TP-4. The study concludes with recommendations for future research in protocol interoperability. Keywords: Defense Data Network, Transport gateways, DDN Internet architecture) Computer protocols, (KR)

- **20. DISTRIBUTION / AVAILABILITY OF ABSTRACT:** [X] UNCLASSIFIED / UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS
- **21. ABSTRACT SECURITY CLASSIFICATION:** UNCLASSIFIED
- **22a. NAME OF RESPONSIBLE INDIVIDUAL:** Allan R. Osborn
- **22b. TELEPHONE (Include Area Code):** (404) 894-3136
- **22c. OFFICE SYMBOL:** ASQBG - C

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

FINAL REPORT

Contract No. B-10-695-SI

US Army Institute for Research in
Management Information, Communications
and
Computer Science Department
Georgia Institute of Technology
Atlanta, GA 30332

PROTOCOL INTEROPERABILITY BETWEEN DDN AND ISO PROTOCOLS

-- A Study and Specification Report --

By

JianYi Tao
Research Associate

Ralph Martinez, Ph.D.
Director

Computer Engineering Research Laboratory
Electrical and Computer Engineering Department
THE UNIVERSITY OF ARIZONA
Tucson, Arizona 85721

August 1988

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF FIGURES

TABLE OF CONTENTS (Continued)

TABLE OF TABLES

1. INTRODUCTION

This chapter gives a brief summary on the background of data communication protocols in the Defense Data Network (DDN) environment and the International Organization for Standardization (ISO) community. It also describes the motivations which stimulated this study task. The objectives of the study task and the structure of the report are also described in this chapter.

1.1 BACKGROUND

In December of 1978, the internetwork protocol suite, The Internet Protocol (IP) and the Transmission Control Protocol (TCP), was recognized and adopted by the Department of Defense (DoD). Since then, it has been well developed in Defense Data Network (DDN) environment and many other academic data networks. The protocol set is well recognized by DDN community as it satisfactorily meets the U.S. military requirements with respect to security, survivability, and reliability. It is also well recognized by the rest of the data communication community as it successfully provides network and transport services to end-users, and the vision of the original architects in the mid-70s is remarkably well-matched to today's data communication need.

On the other hand, the International Organization for Standardization (ISO), especially the Technical Committee 97 in ISO, along with some other standards organizations such as the International Telephone and Telegraph Consultative Committee (CCITT), have been developing high-quality standards for networking services, protocols, and interfaces for Open System Interconnection (OSI) in years. It is felt even more urgent in the past few years, with the rapid evolution of communication technologies and the system architectures, to the construction of distributed application environment with more advanced data communication standards, by a very large community of computer system builders. Some resulting standards have already been recognized as international standards. The ISO IS 8473 as a Network Layer protocol and ISO IS 8072/8073 as a Transport Layer protocol are two examples. In the United States, these protocols and some other layer's protocols have been adopted with minor modifications by the National Bureau of Standards (NBS) for some governmental organizations, and implemented by several vendors on a few systems. The ISO protocols are more favorable in the European countries, such as in the NATO member countries.

Under these circumstances, the critical problem of interoperability has arisen in interconnecting those independent DoD communities with other federal, intelligent, security, and commercial agencies in U.S., with other international organizations, and with the rest of NATO. This research is addressed to study the interoperability issues between the DDN protocols (TCP/IP) and the OSI protocols (TP-4).

1.2 MOTIVATIONS

In 1985, under the request by the DoD and NBS, the Committee on Computer-Computer Communication Protocol in the National Research Council did a transport protocol study of the DoD TCP/IP and the ISO Transport Protocol Class 4 (TP-4) [NRC 85]. The result shows that both protocols are functionally equivalent, both provide essentially similar services, and both are sufficiently equivalent in security-related properties. This study recommended with three alternative approaches for the DoD to solve the interoperability problem:

1) Immediately specify the TP-4 as a costandard -- for newly added systems to reduce the transition cost later.

2) Announce the intention to have TP-4 as a costandard -- to wait for the satisfactory demonstration of the protocol and implementation suitability; and to wait to use commercial products with low developing cost.

3) Continue to use the TCP, delay the adoption of the TP-4 -- to avoid faulty system behavior and unnecessary delay.

When more than one protocol is concerned in the data communication community, there must be the compatibility problem to be faced before and after the new protocol is adopted, until all the protocols are truly unified with unique standard protocol set. This is one of the reasons which stimulated this study.

Meanwhile, it is NATO's intention to introduce ISO protocols as far as possible in all NATO's new systems, some valuable research efforts were performed in the SHAPE Technical Center, as a joint effort with U.S. DoD, in the area of conversion between TCP and ISO transport protocols [GROEN 86]. The direct conversion method was studied as a method of achieving interoperability between the NATO and DDN data communications systems. However, the approach imposes quite a few restrictions on the transport service users to resolve the mismatches. So the second reason for this study is to find some alternative approaches to relax the restrictions on the transport service users.

Started in 1986, the Network Working Group in ARPANET proposed in RFC 1006 to implement ISO Transport Protocol Class 0 (TP-0) services on top of the TCP. The basic idea is that the TCP/IP protocols provide the error-free virtual circuit connections between transport service users so that only TP-0 of the ISO transport service is needed to interconnect DDN with the ISO protocol world. This approach could help the transition from the TCP/IP protocol suite to the ISO protocol suite. But there are some other issues which need to be further discussed. This is the third reason for the study task.

The pace of implementing ISO TP-4 is relatively slow, while the interests in TCP/IP increased dramatically recently. IBM and some research institutes in European countries jumped to TCP/IP in last six months. A study result released in December, 1987 shown that 56 percent of those surveyed are still willing to stay on TCP/IP with no plan to migrate to ISO. The implication of this survey means that it is going to be possible to have TCP/IP networks that have ISO protocols running simultaneously with TCP/IP protocols.

The ISO protocols have long been recognized as the trend of the protocol development. While the TCP/IP protocols have already occupied a large market, it will be inevitably required to have the well-developed protocol conversions between the DDN TCP/IP and the ISO TP. This is another reason that the protocol conversion could be quite important in the near future.

1.3 OBJECTIVES

This study is focused on the following problems:

1) The general issues involved in the protocol conversion;

2) The protocol conversion in the DDN to ISO environment;

3) More detailed understanding of both protocol suites;

4) Approaches to achieve the interoperability between TCP/IP and TP-4;

5) Proposal to the future studies in the area.

1.4 STRUCTURE OF REPORT

The report is divided into four major sections. Part I contains discussions about the interoperable protocols environment in DDN in the future. Part II and III are devoted to the DDN and ISO protocol suites. Part IV is for the protocol conversion between the TCP/IP and the TP-4.

PART I. INTEROPERABLE PROTOCOLS IN DDN ENVIRONMENT

In this section, the major issues in the protocol conversion are presented. The protocol interoperability requirements in the DDN environment are discussed. The protocol conversion tasks are defined to meet the requirements.

2. MAJOR ISSUES IN THE PROTOCOL CONVERSION

During 1987 there was a lot of controversy in the development gatewaying strategies. Since it is inevitable to have heterogeneous networks coexisting in the data communication community, the protocol conversions will be a permanent fact of life. For example, today we are concerned about the gatewaying between DDN TCP/IP community and the ISO/OSI protocol communities in the same digital data communication world. Soon we will find ourselves concerned with the gatewaying between the global telephone network with integrated services (ISDN) and the global Wide Area Network with digital data communication (GWAN). Some experts are arguing that the ISO/OSI protocol architecture should be considered as a worldwide standard architecture toward which all existing architectures should converge in the long term. Other experts argue that only an intermediate protocol for the protocol conversions in the near term should be considered.

The significance of the gateway design for protocol conversion is based on the principle that ALL the protocol layers above the gatewaying layer must be compatible. This rule should be strictly reserved in the gateway design. The gateways implemented at the network layer (DDN IP and ISO IP) are of limited general use in the long term. Some of the arguments indicate that gatewaying should occur at or above the transport layer. And some arguments also suggest that the best approach is to perform gatewaying exactly at the transport layer to minimize the gatewaying effort.

Gateways can be implemented at the transport protocol peer or at the transport service access point. When this approach is used, how should the end-to-end reliable service property at the transport layer be maintained? How should its significance be evaluated in the military environment where the survivability is critical important? When the service primitive is not provided by one of the converted protocols, should it become the restriction imposed on the service users? Should it be resolved by the protocol conversion mechanism in the gateway as much as possible? Should more intelligence in the conversion be considered to handle the mismatches? Should direct packet-to-packet conversions be preferred for higher throughput with more restrictions on the user side? The last question is how the low throughput and performance should be improved for heavy duty gateways? These are questions which must be addressed in this research.

3. PROTOCOL INTEROPERABILITY REQUIREMENTS IN DDN

It is a well-recognized trend in the data communication protocol development to converge to the ISO standardized protocols. As suggested by the report from the transport protocol study group in the National Research Council, the DDN community should be ready to have the protocol convergence in certain future time period [NRC 85]. On the other hand, the delayed availability of the ISO TP-4 implementations is causing more user groups jumping over to the TCP/IP internetworking protocols. The effect of the current situation will definitely put more importance onto the protocol conversion task when the conversion is actually needed.

3.1 Interconnecting DDN and ISO Networks

This is the basic problem of interconnection between two network systems, such as the TCP/IP-based DDN in the U.S. and the ISO based network in the rest of NATO. Figure 3.1 shows this internetworking scenario. Only one set of protocol suite is implemented in each network in this case. The gateway is required to interconnect the two networks.

This kind of interoperability is always required before any protocol transition or convergence plan is made.

3.2 Interconnecting TCP and ISO Users in the Same Network

This is the problem that both the TCP/IP and the ISO protocols coexist in the same network as costandards. The network here could be a set of interconnected subnetworks. Two protocol suites can coexist in any subnetworks. The users using different protocol suites are communicated through the intermediate gateways, as shown in Figure 3.2. The DDN Internet is an example of the case.

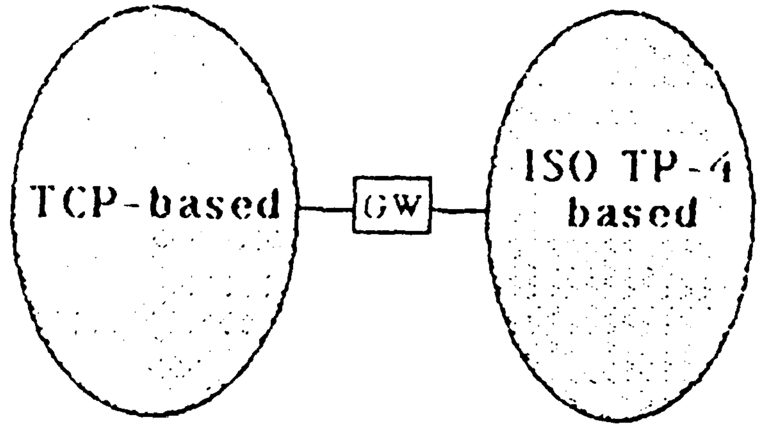


Figure 3.1 Interconnecting the TCP-based Network with the ISO-based Network

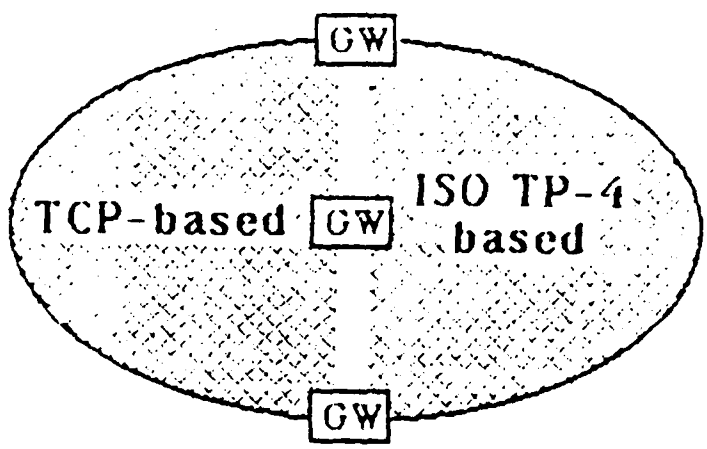


Figure 3.2 Interconnecting TCP/ISO Users In the Same Network

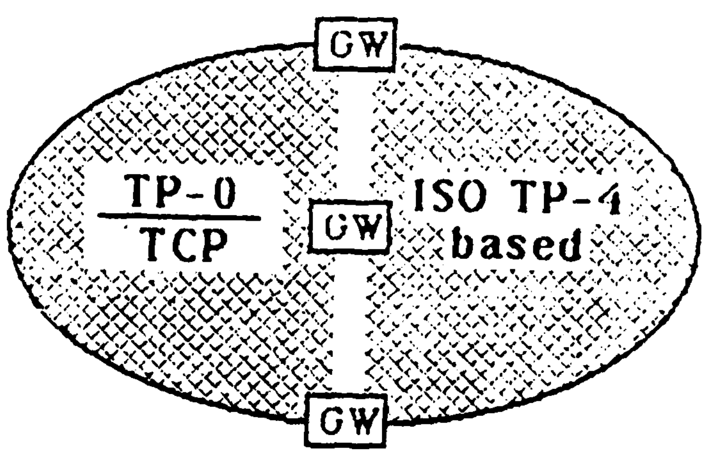


Figure 3.3 Interconnecting TP-0/TCP users with the Rest

3.3 Interconnecting TP-0 Users with the Rest

This is the case where ISO TP-0 services are implemented on some nodes above the TCP/IP protocols, as proposed in [ROSE 87]. As higher level services and protocols above the transport layer are usually implemented as user-callable utilities on the host computers, it is desirable to offer them directly in the DDN Internet now without disrupting existing facilities. This will permit DDN users to develop expertise with ISO applications while there is still a lot of work being done to get good implementations of ISO transport/network layers. It will also permit more graceful convergence and transition strategy from TCP/IP networks to the ISO-based networks in the medium and long term. The interconnection model is shown in Figure 3.3. This migration strategy is based on the notion of gatewaying between the TCP/IP and ISO protocol suites at the transport layer.

4. PROTOCOL CONVERSION TASKS

The problem of protocol conversion will be discussed from three aspects: general gateway functions, gateway functions required in the DDN environment, and gateway functions attended in this study task.

4.1 GENERAL GATEWAY FUNCTIONS

Internet gateways, regardless of their application environment, must perform a variety of functions in order to make data communications between different networks compatible [MART 87b]. The functions of generalized internet gateways are discussed here:

1. Medium Transformation - A gateway must translate messages between different transmission media,such as LAN RF broadband or baseband digital signals, and the serial 1822 or X.25 interfaces of the DDN packet switching nodes. Signaling schemes to each network must be present in the gateway.

2. Media Access Translation - The media access schemes on the LAN side of the gateway must be present in the gateway. Media access schemes on LANs, such as CSMA/CD or token passing 802.4, must be present in the gateway. Access schemes to the DDN must also be present.

3. Address Translation - Network addressing schemes are

different on each network, so that the gateway must perform address translation. For example, the IEEE 802.3 LAN uses a 48 bit flat addressing scheme and the DDN uses a 32 bit two-level addressing scheme. The gateway must recognize internet addressing schemes when interconnecting multiple networks.

4. Protocol Transformation - The network protocols of each network must be transformed through decapsulation and encapsulation steps in L part of the gateway. For the DDN, the Internet Protocol (IP) and the Transmission Control Protocol (TCP) must encapsulate to a LAN message. The LAN protocol headers must be stripped before hand-off to the TCP/IP protocols. In the case of internet environment, a gateway-to-gateway protocol must be implemented.

5. Message Buffering and Flow Control - The gateway must be able to buffer messages from each network and flow control the network interfaces when the buffers are full. The flow control mechanisms buffer sizes are critical to the performance of the gateway.

6. Reliable Connection Management - The gateway must provide an error free link between two end-users on the networks by adhering to the error control and re-transmission mechanisms in the network protocols. The status of the connection must be made available to the user when error conditions arise.

7. Fault Detection and Reporting - The gateway must be able to detect connection status when establishing and maintaining a connection between two end-users. The gateway then reports to the users the condition of the links, gateways, and networks in the connection path, if a problem should occur.

8. Performance Monitoring and Statistics - The gateway must be able to monitor its performance relative to packet throughput and network routing statistics. These parameters can be read locally or remotely from the gateway and used for internet management.

9. Security Control Mechanisms - The gateway must adhere to internet security control and management procedures. This might include generation and routing of encryption keys and cryptograhic algorithms.

10. Real-Time Response - The gateways must process packet traffic from the networks in real-time so that user response times are not compromised. The gateway must accommodate the differences in network response times. Real-time response is also important during interactive user sessions. The gateway must sustain the communication rates of each network.

11. Parallel Processing Architecture - The gateway must contain parallel processing architecture to sustain the network transmission speeds. Dedicated protocols and communication modules must exist to achieve the performance throughput

required by connection to multiple networks.

12. ISDN Interfaces - Gateways must eventually interface to integrated services digital networks (ISDNs) for data, voice, and video communications. The gateway must interconnect to ISDNs and their predecessors.

13. Multiple Network Interconnection - Gateways in distributed C3 systems must have the interfaces and link parts to access multiple communications systems and networks.

14. Multi-Level Security and Key Distribution - Communications between end-users in the distributed C3 system will require multi-level security for sensitive information and the gateway must preserve the data security characteristics of several networks.

15. Dynamic Network Topology Reconfiguration - Since the gateways are interconnected to multiple networks it is feasible to use the gateway to keep network status and give this information to the network management function for reconfiguration when nodes and networks fail or are destroyed.

16. Network Reachability - The gateway must determine the reachability and availability of neighboring networks. Network status must be exchanged between gateways so that alternate network hops be taken when a path is down.

17. Internet Management and Control - The gateway interacts with an Internet Management and Control Center to assist in the daily operation and reliability of the networks. The gateway performance monitoring function collect performance data and presents it to the Control Center.

4.2 GATEWAY FUNCTIONS BETWEEN DDN AND ISO/OSI

4.2.1 Differences between Network and Transport Layer Gateways

Figure 4.1 and 4.2 show the major difference between the network layer gateways (IP gateway) developed in the previous research work and the transport/network layer gateways in the current researches [MART 87a]. In the IP gateway case, it is concentrated on the connectability up to the network layer. The medium transformation, the media access translation, and the subnetwork protocol translation are performed in the IP gateway, while the same Internet Protocol and the same upper layer protocols are used. Gateways at the Network Layer are called

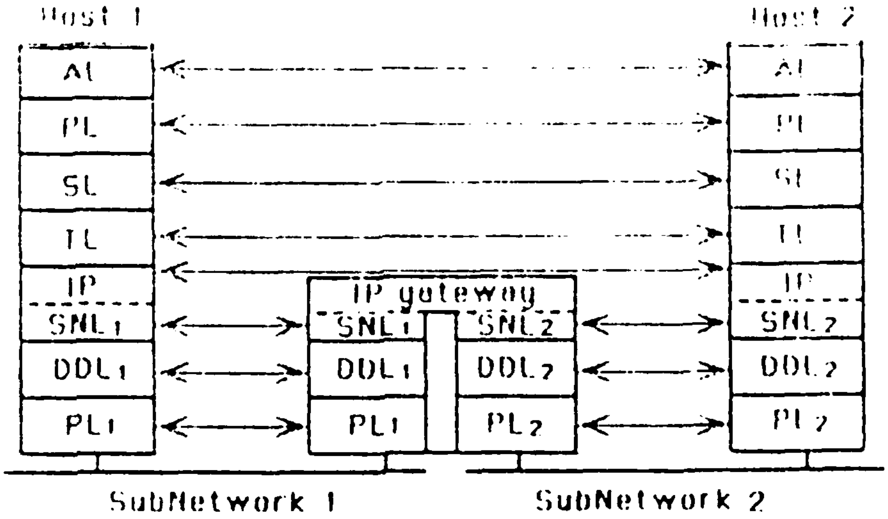


Figure 4.1 **The IP Gateway Model**

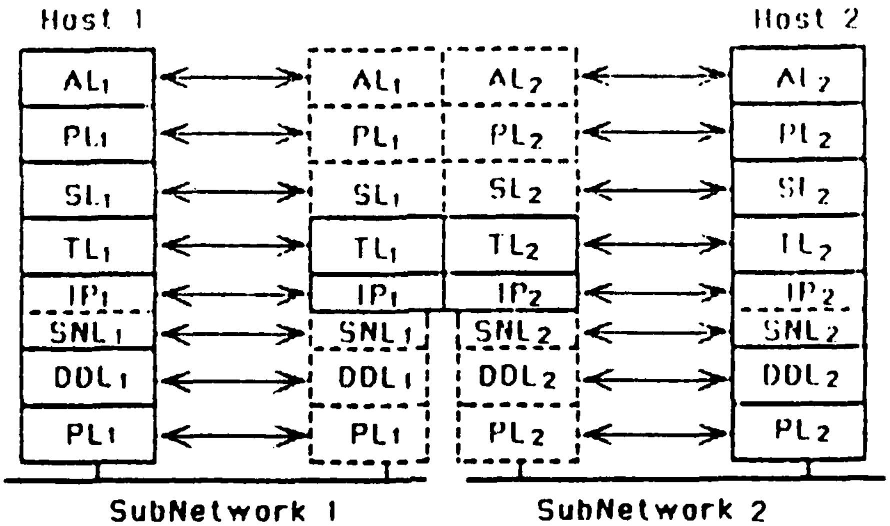


Figure 4.2 **The Model of Transport/Above Gateway in Concern**

Routers [MART 87b].

In the second case, more complicated functions are performed in gateway than those in the router case. As a result, it increases the complexity of the gateway design. While the gatewaying at the higher layers (Session, Presentation, and Application layers) are optional, depending on the case concerned. These situation will be described in the following sections.

4.4.2 Three Cases of Transport Layer Gateways in DDN

The cases discussed in following section can be applied to both situations explained in Section 3.1 and 3.2: interconnecting DDN and the ISO based networks, and interconnecting TCP/IP users and ISO protocol users within the same network. This is based on the fact that the differences at the lower layers up to the network layer are not important issues here.

Case one implements not only the Transport/Network layers, but also other upper layers, as shown in Figure 4.3. The gateway itself actually becomes a node in both networks, performing store-and-forward functions between application layer protocols. This kind of gateway will be able to connect the TCP/IP based DDN world with the ISO protocol based networks without changing anything in the DDN.

The advantages of this approach are:

1) the least effort is needed by the user nodes in the DDN side;

2) the gateway implementation could be simple using a host computer to be connected to both networks, running both sets of protocol separately in multi-tasking environment, performing message translation at the user level; or performing the application protocol conversion at the application layer [MART 87a,b].

The disadvantages of this approach are:

1) lower throughput in the gateway,

2) longer node delays,

3) low performance when used for the interactive message transferring,

4) that the gateway becomes the single point of failure in the

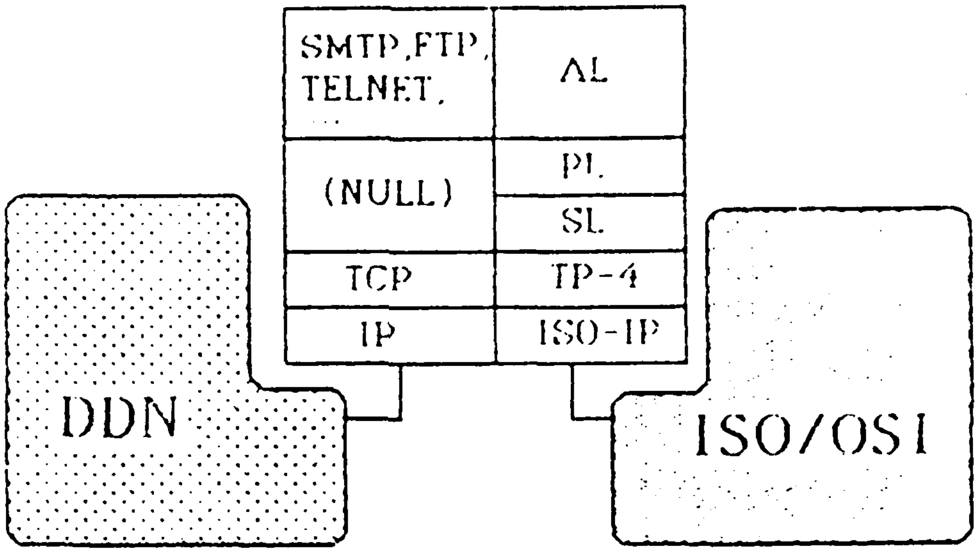


Figure 4.3 **Gateway with Upper Layer Protocols Implemented (Case One)**

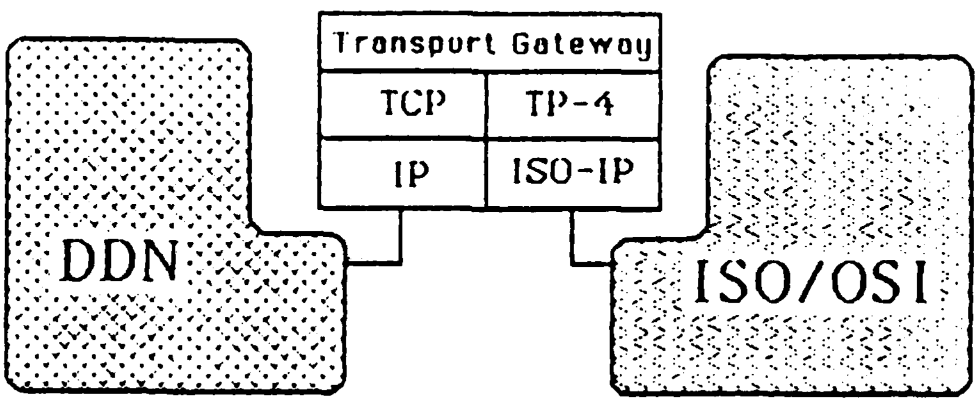


Figure 4.4 **Transport Layer Gateway without Upper Layers (Case Two)**

DDN environment where the survivability is critical.

Case two implements the protocol conversion at the network and the transport layers, as shown in Figure 4.4. This approach requires the ISO upper layer protocols being implemented on those DDN nodes whose users want to communicate with the users in the ISO based networks.

The advantages of the approach are:

1) If the protocol conversion is implemented at the protocol peer entity at the transport layer, the end-to-end property of the connection between the end users can be sustained. This is very important in the DDN environment for survivability;

2) This is more reasonable approach for protocol convergence in the future.

The disadvantages of this approach are:

1) The implementation of the gateway is more complicated due to the complexity of the protocol at the transport layer;

2) Only those users with ISO upper layer protocols implemented in the node can communicate with other nodes.

Figure 4.5 shows the model of Case Three. The TP-0 protocol at the top of the TCP is considered in this approach, but it is not necessary the only choice. Actually, it is considered as a general study. The different classes of ISO Transport services are not compatible by the nature. Gatewaying is needed to get the users with different classes of transport services communicated and could be quite easily extended to the actual situation in real need. One example could be the connections to long haul networks implementing LAN protocols. As discussed in some papers, this approach is considered to reduce the overhead of overly complicated protocols.

4.2.3 Proposed Transport Gateway with Multiple Functions

Putting the three models from previous discussion together, a multiple-function gateway is discussed in this section. The multiple function transport gateway structure is shown in Figure 4.6.

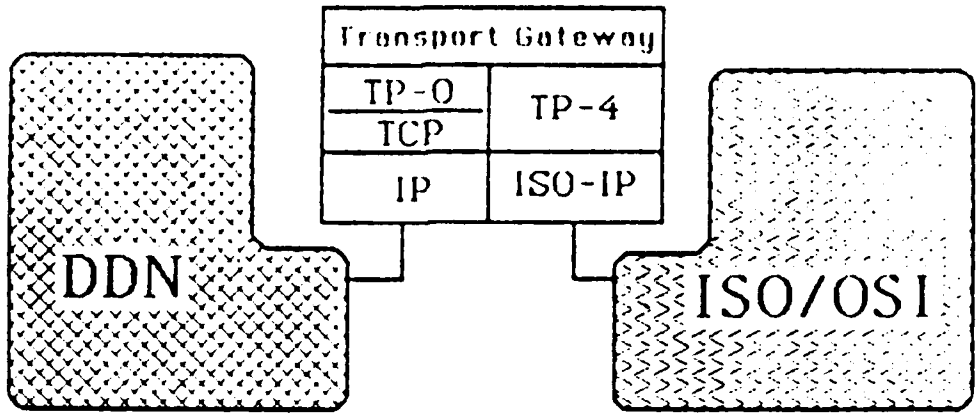


Figure 4.5 Transport Layer Gateway with TP-0 on Top of TCP
(Case Three)

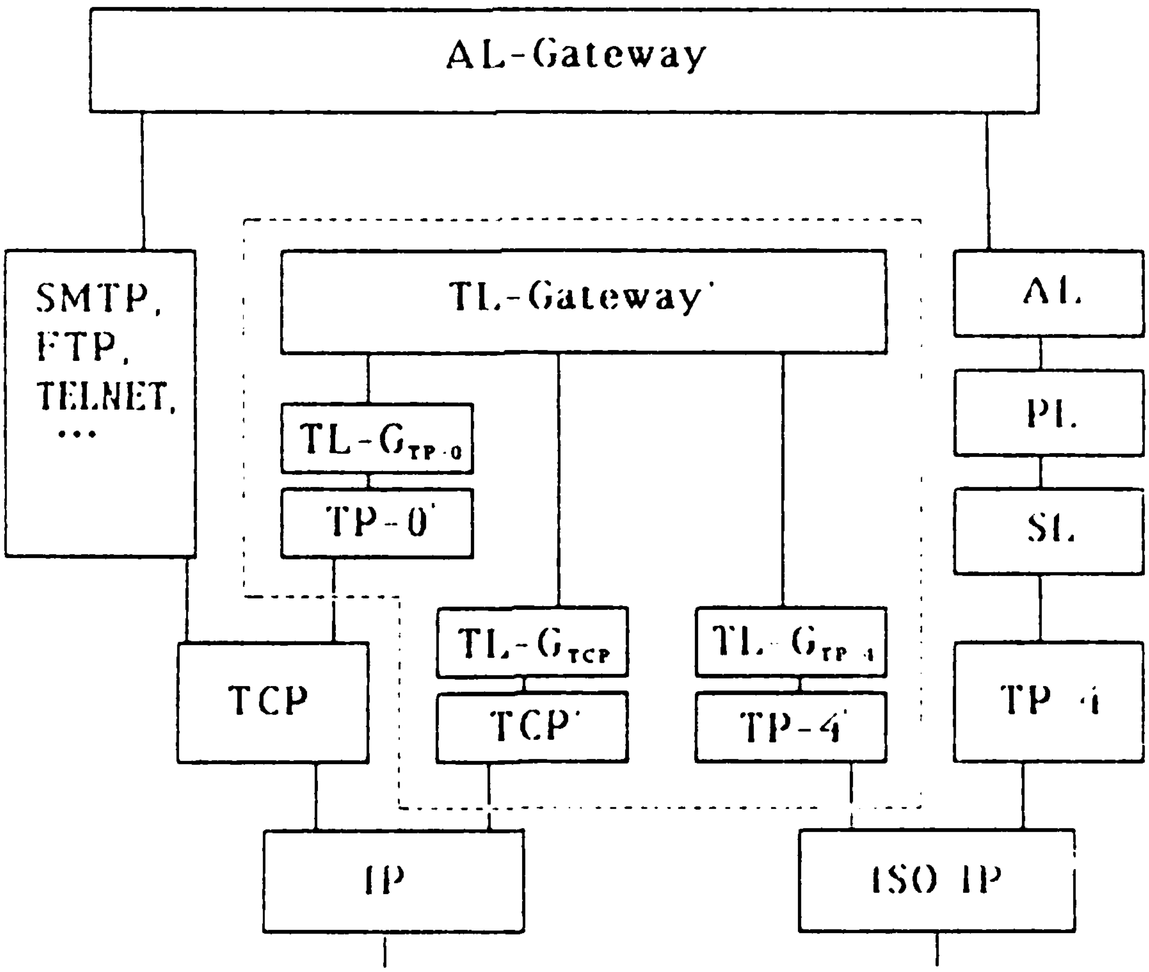


Figure 4.6 Multiple-function Transport Gateway

Following notations are used in the Figure 4.6:

IP: DDN Internet Protocol
TCP: DDN Transmission Control Protocol
SMP,FTP,TELNET: DDN Application Layer Protocol

ISO IP: Equivalent Internet Protocol in ISO (CLNS, 8473)
TP-4: ISO Transport Layer Protocol, Service Class 4
SL: ISO Session Layer Protocols
PL: ISO Presentation Layer Protocols
AL: ISO Application Layer Protocols

AL-Gateway: DDN-ISO Gateway at/above Application Layer

TL-Gateway': Transport Gateway, link submodule
TL-G TCP : Transport Gateway, interface submodule to TCP
TCP': Transport Gateway, TCP submodule
TL-G TP-4 : Transport Gateway, interface submodule to TP-4
TP-4': Transport Gateway, TP-4 submodule
TL-G TP-0 : Transport Gateway, interface submodule to TP-0
TP-0': Transport Gateway, TP-0 submodule

The AL-Gateway module in the figure performs the gatewaying at/above the application layer, in order to interconnect the upper layer protocols between DDN and ISO world. The protocol conversion in this model is discussed as Case Three in the previous section.

The Transport gateway module is made of several submodules. To interconnect different transport layer protocols, a general link submodule is used to perform gatewaying tasks independent of particular transport protocols. The protocol-dependent tasks, such as checksum production, reproduction, and error detection, are performed in the protocol-specific submodules. The gaps between the protocol-specific submodules and the general link submodule is matched up by the interfacing submodules. The mismatches between the interconnected protocols will be notified to both sides. Some of the conflicts are handled in each interfacing submodules, and some others are handled in the general link submodule. Some decision-makings are required in the link submodule. Another phase of research determined the functions which are candidates for protocol negotiation in the gateways.

4.3 GATEWAYING CONCERNED IN THIS STUDY

This study examines the issues and design approaches related to transport layer gateways. The gateway functions are performed by the TL-G submodules in Figure 4.6 (surrounded by the dash lines). The protocol conversion tasks and the conversion algorithms will be specified in the last part of the report.

5. FUNCTIONAL ELEMENTS CONCERNED IN THE TRANSPORT GATEWAYS

This chapter presents some explanations and definitions to those terminologies related to the functional elements in transport and network layers, based on the OSI model. Special issues are also discussed for those functional elements which requires special processing in the transport gateways.

5.1 CONNECTION MANAGEMENT

Connection at N-layer is an association established between the N-layer peer entities to be used by the users above the N-layer. The peer entities in the connection are identified by their addresses, or service access points. Usually, the N-layer protocol user initiates the N-layer connection establishment. The complexity of the connection establishment operation at different layers is dependent on the specific layer. The connection establishment at the N layer requires that the (N-1) layer connection is available and both N-layer peer entities are in the state in which they can execute the connection establishment protocol exchange. Otherwise, some manipulation should take place to handle the exception. There are different options in the connection establishment. In some protocols, three-way handshaking are used. Some protocols allow data transferring by the connection establishment protocol exchange.

The release of the N-connection is normally initiated by one of user above the N-layer. It may also be initiated by one of the peer entities at the N-layer as a result of an exception occurred at the N-layer or the layer below. The user data may be lost upon this condition. The orderly release of the N connection requires protocol exchanges between the N-layer peer entities. One example is the common reference to time in the connection release.

The connection establishment at the transport layer is more complicated functional element, and quite versatile between different transport protocols. Special cares are required in the transport gateways. For example, when the connection establishment is initiated by the TCP user through the gateway to the TP-4 user at the other side, the transport gateway needs to act as the TCP peer entity and the TCP user in order to get the connection established and to get complete information from the initiating TCP peer entity. It also needs to act as the TP-4 user and TP-4 peer entity in order to establish the connection to the destination TP-4 user. Since the transparency should be available during the operation, more intelligence is required in the transport gateway. Another example is to resolve the difference in the addressing. The solution to this problem can be very complicated, depending on the addressing schemes involved.

5.2 DATA TRANSFER

User data and control information are transferred between peer entities using protocol-data-units (PDUs). Several functions are required in the gateway for the operation.

An expedited PDU is a service PDU which is transferred and processed with higher priority over that of normal PDUs. It is used to transfer small amount of data infrequently, such as for signaling and interrupt purposes. An expedited PDU is transferred independently from flow control over the normal data, and is guaranteed to be delivered before any subsequent normal PDU or expedited PDU can be sent on the connection.

Special care is needed in the gateway design for the data transfers. For example, TP-4 allows to piggybacking user data on a connection establishment, while TCP does not support this option. TP-4 supports the expedited data transfers, while TCP supports the PUSH option in the normal data transfers.

5.3 FLOW CONTROL

Flow control is the function which controls the amount of data flowing within a layer or between adjacent layers. The flow control which regulates the data rate between peer entities at the same layer is called peer flow control. The flow control which regulates the data rate between N-entity and (N-1) entity is called layer interface flow control. There exist several flow control mechanisms, and the "sliding window" is one such technique.

5.4 SEQUENCING

Sequencing is the function which preserves the order of PDUs being transferred between peer entities. It is required when the received data may not be guaranteed to be in the same order as they are delivered. Sequencing usually requires additional protocol control information.

5.5 MULTIPLEXING AND DEMULTIPLEXING

Multiplexing at the N-layer is the function by which more than one N-connections are supported by a single (N-1) connection. As protocol data are multiplexed by the sender, demultiplexing should be performed by the receiver to get the protocol data recovered for different connections.

5.6 SPLITTING AND RECOMBINING

Contrary to the multiplexing, splitting at the N-layer is the function that one N-connection is supported by more than one (N-1) connections. As protocol data are splitted at the sender, recombining should be performed at the receiver to recover the protocol data.

5.7 SEGMENTING, BLOCKING, AND CONCATENATION

Segmenting is the function at the sender which maps one data unit into multiple data units, or packets. Reassembling is the reverse function of segmenting on the receiver side. Blocking is a function which maps multiple data units into one data unit at the same layer. Deblocking is performed at the other side as the reverse function. Concatenation is a function which maps multiple N data units into one N-1 data unit. Separation is required at the other side as the reverse function.

5.8 ROUTING

Routing is the function which enables communication to be relayed by a chain of entities. The routed communication may be transparent to both higher and lower protocol layers. Most of routing function is performed at the network layer. The routing function is a special important issue in the network layer gateway design for internetworking environments. Routing algorithms must account for differences in internet addresses.

5.9 ERROR HANDLING

Error handling function may be performed in several ways in the OSI layers. An acknowledgment mechanism may be used to obtain a higher probability of detecting data unit loss. The data-unit is made uniquely identifiable, so that the receiver can inform the receipt or nonreceipt of the data unit. The sender may take remedial action accordingly. Usually, the acknowledgment mechanism may require additional protocol control information. Other error detection and notification functions can also be used for the similar purpose, such as "checksum" and "frame check sequence" operations to detect the errors in the received data-units.

5.10 RESET

Reset is a function which sets the corresponding peer entities to a predefined state after some uncorrectable error conditions occur, such as the loss of synchronization. It may cause a possible loss or duplication of data, however, all protocol states are eventually reset.

5.11 MANAGEMENT

There exist three aspects of management in the OSI model. The Application-management, in the application layer, is related to the management of OSI application processes. The system-management relates to the management of OSI resources and their status across all layers of the OSI architecture. The layer-management relates to the management of specific layers, partly performed in the layer such as activation and error control and partly performed as a subset of system-management.

The following sections describes related protocols in the DDN environment and ISO/OSI architecture, based on the definitions described in this chapter.

PART II. THE DDN PROTOCOL SUITE

This and the following chapters explain the DDN internetwork protocol suite in detail, with the emphasis on the TCP and IP protocols. Readers who are knowledgeable in the DoD protocol suite can skip the sections on the internal protocol mechanisms. For readers, not versed in this area, the following sections offer a background required to understand the DDN and ISO protocols.

6. DDN INTERNET ARCHITECTURE

Besides the common requirements for data communication networks, the DoD has its own special requirements for the DDN internet architecture:

1) The security concern with data security and communication security;

2) The survivability concern with minimization of critical control nodes;

3) The reliability concern with adaptive, robust data distribution.

Accordingly, the DoD adopted the internetworking policy to implement the Transmission Control Protocol (TCP) on each host in the internet for the end-to-end reliable transport protocol. The Internet Protocol (IP), the Gateway to Gateway Protocol (GGP), and the Exterior Gateway Protocol (EGP) were defined for the intra- and inter-network communication protocols. The DDN Internet architecture and its protocol suite are shown in Figure 6.1. Following sections describes the IP and TCP protocols in detail. The GGP and EGP will not be discussed in the case that they are not closely related to this conversion task.

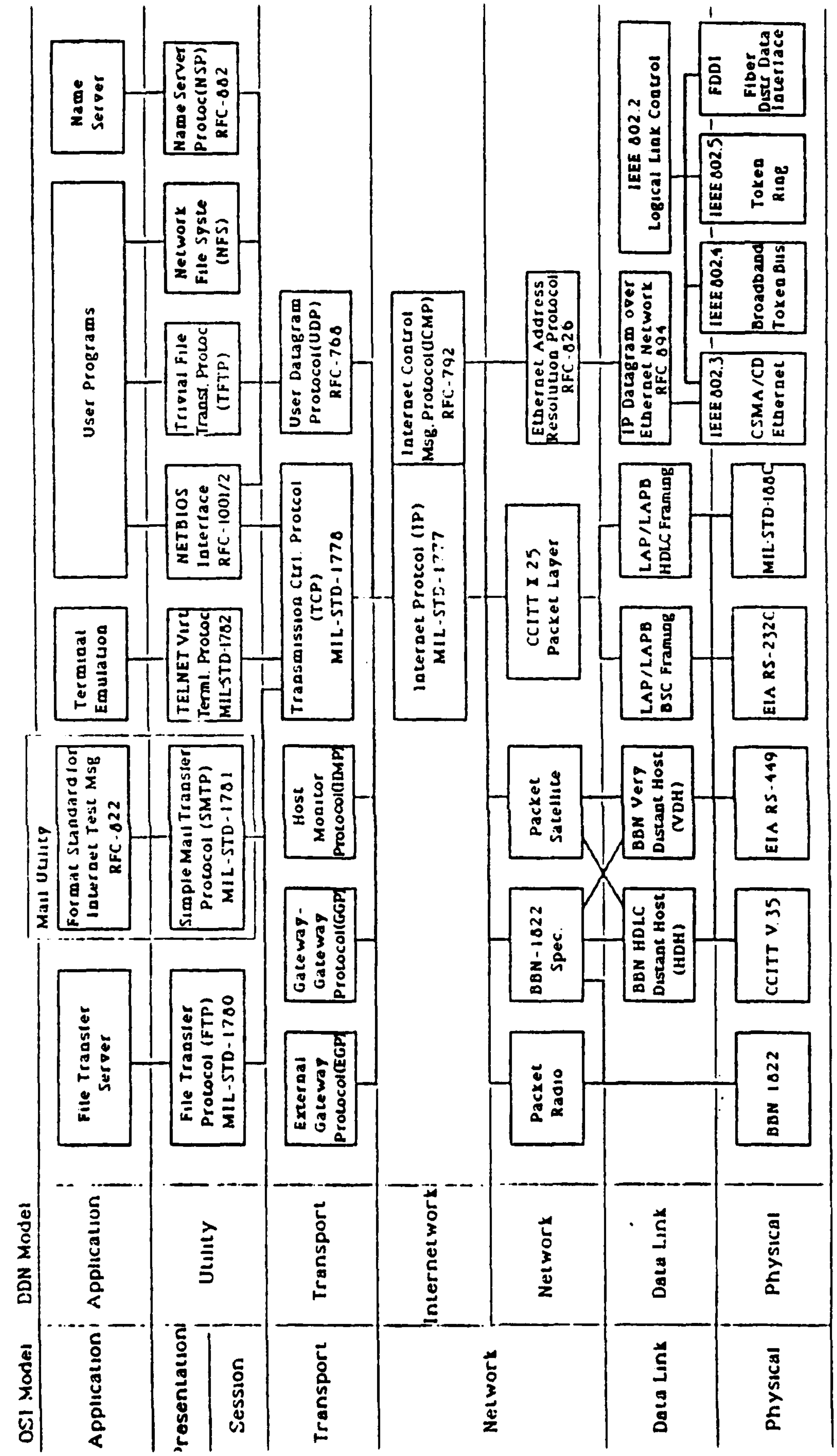


Figure 6.1 DDN Internet Architecture

Figure 6.1 DDN Internetwork Architecture

7. INTERNET PROTOCOL (IP)

The Internet Protocol (IP) provides the basic service for transmitting datagrams between source and destination hosts. It was located at the "Gateway" level in original DDN Internet architecture, and now it is located in the "Internetwork" sublayer as shown in Figure 6.1. The IP service is provided based on the various local network services underneath in the "Network" sublayer.

1) **Datagram Transfers.** The datagrams are blocks of data, or protocol data units (PDUs) in OSI Reference Model. In the DoD internetwork environment, a datagram is an independent data entity unrelated to each other. There are no connections or virtual circuits established. The PDUs are sent to the lower layers and no indication is received of their reaching the destination. The services for reliability in the data transmission are provides by higher layer protocol (e.g., TCP) above the Internet Protocol.

2) **Addressing.** The IP module resides in each network host and each gateway interconnecting networks. The source and destination addresses of the hosts are identified by a fixed length address field in the header of datagram (32 bits). The IP modules along the path transmit the datagrams from the source host through the intermediate hosts and gateways to the destination while making routing decisions to select the transmission path.

3) **Fragmentation and reassembly.** The IP module modifies the size of the datagrams by breaking the datagrams to small lengths and putting them back together later at the destination. Ip does this when it is necessary to transmit long datagrams through "smaller packet" networks.

4) **Types and qualities of service.** It is a generalized set of parameters characterizing different service choices, to be used by gateways to select the actual transmission parameters for a particular network.

7.1 OPERATIONAL MODEL DESCRIPTIONS

The IP provides datagram service to upper layer protocols such as

TCP. It calls for services from Network layer and lower layers, depending on the networks connected. Detected errors at the IP layer are reported via the Internet Control Message Protocol (ICMP) which is required to be implemented with IP.

The data from the upper layer protocol module is passed to the local IP module with the destination address and other parameters as the arguments. The IP module prepares a datagram header with those arguments and attaches the data to it. It also determines, according to the destination address, the local network address for the immediate receiver (network node or gateway). The datagram is passed to the local network interface module. The local network interface module, in turn, creates its own packet, attaches the datagram to it, and sends it over to next immediate receiver. As the packet is received, the local network packet header will be stripped off, the datagram will be handed over to the IP module. That IP module will determine whether the datagram reaches the destination node. If not, it will decide how the datagram is to be forwarded to next receiver by the destination address. The operations will be repeated until the datagram reaches its destination. If the datagram reaches its destination, the IP header will be stripped off, and the data will be passed over to the upper layer protocol module.

The upper layer protocols may use logical names to indicate the destination host address. And mapping are performed as:

Protocol level	Mapping task
Upper levels	logical name -- Internet address
IP level	Internet address -- local net address
Lower level	local net address -- routes

It should be noticed that one host can have several physical interfaces (multi-homing) to local network, or a single physical host can use several distinct internet addresses, acting as if it were several distinct hosts.

The fragmentation takes place more than once along the path as necessary. When it happens, the data of the long datagram is divided into portions on a eight-byte boundary, and the datagram header will be duplicated for each new, shorter datagram. The total length field in the headers is adjusted to current actual length. The more-fragment flag of all but last datagram is set to one, and is copied from long datagram over to the last shorter datagram; the fragment offset field of all shorter datagrams is set to the sum of the original value of long datagram and the offset value among shorter datagrams, to ensure the correctness for multiple fragmentation. To assemble the fragments correctly, the data from datagrams with same identification, source and destination addresses, and protocol field values will be arranged and combined according to the fragment offset field of each datagram.

7.2 PROTOCOL SPECIFICATION

The datagram header format of the IP is described in Figure 7.1.

Version (4 bits): The IP version is 4.

Internet Header Length (IHL, 4 bits): The length of datagram header in 4-byte words. The minimum IHL is 5 without any options. The maximal internet header is 60 bytes.

Type of Service (8 bits): These are parameters for the service choice, to specify the treatment of the datagram during its transmission through the system, as explained in Figure 7.2. It is actually a three way tradeoff between low-delay, high-reliability, and high-throughput. A 3-bit precedence field is included.

Total Length (16 bits): The length of the datagram *in* bytes. All hosts must be prepared to accept datagrams of up to 576 bytes.

Identification (16 bits): The identifying value assigned by the sender to help in assembling the fragments of a datagram.

Flags (3 bits): The control flag for fragmentation, as described in Figure 7.3.

Fragment Offset (13 bits): The offset location of this fragment in the assembled datagram, indicated in 8-byte words. The fragment offset for the first fragment is zero.

Time to Live (8 bits): The maximum time limit in seconds the datagram is allowed to remain in the internet system. It is set up by the sender, decreased by every IP module along the path, and the datagram is self-destructed when it becomes zero.

Protocol (8 bits): This field indicates the next level protocol used in the data portion of the internet datagram.

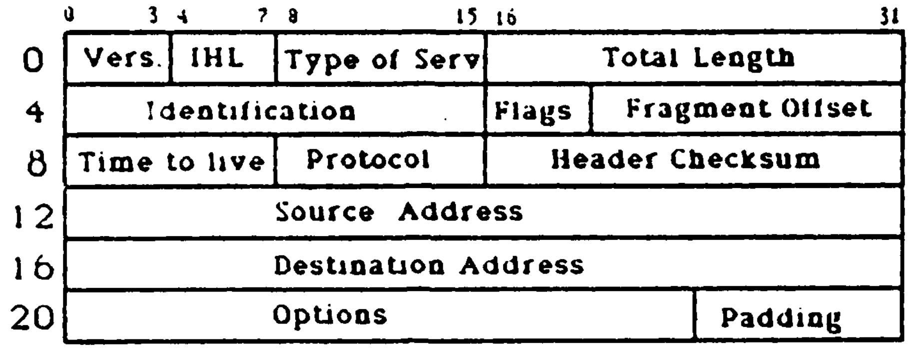


Figure 7.1 IP Header Format

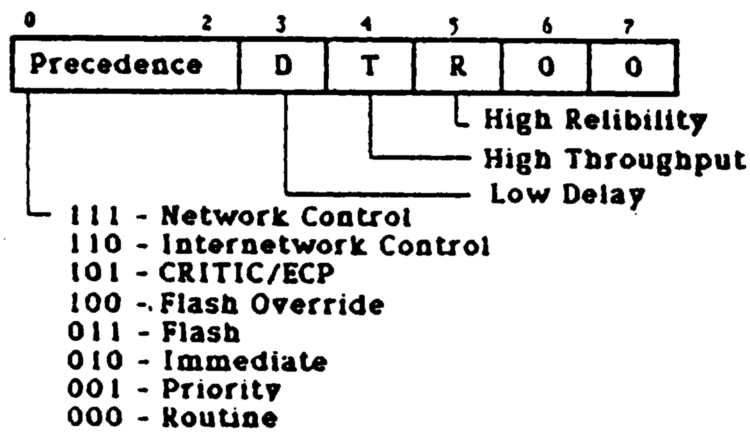


Figure 7.2 Type of Service in IP

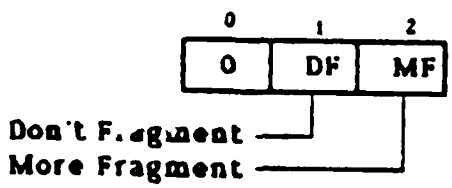


Figure 7.3 Flags in IP

Some of the examples are listed in Table 7.1.

Header Checksum (16 bits): It is one verification of correctness in the transmission. The checksum is on the header only, calculated as the 16-bit one's complement of the one's complement sum of all 16-bit words in the header. The value for this field is zero for the calculation. It is recalculated and verified at each point that the internet header is processed.

Source Address (32 bits)

Destination Address (32 bits)

There are three classes of internet addresses, as illustrated in Figure 7.4. There classes handle the different cases underlying subnetwork types.

Options (in variable length): It is another field that the IP users can use to specify the special treatment for the datagram, useful in some special situations, including timestamps, security, and special routing. Some examples are displayed in Figure 7.5.

7.3 INTERFACE DESCRIPTION

IP provides two kinds of service calls for upper layer protocols. But the actual implementation is more or less system dependent.

SEND (src, dst, prot, TOS, TTL, BufPtr, len, id, DF, opt) ==> result

Upon unsuccessful service calls, such as bad arguments, unaccepted datagram by local network, a reasonable report must be returned, such as the cause of the failure, to the IP user. The details of the report are up to the implementation.

RECV (BufPtr, prot) ==> (result, src, dst, TOS, len, opt)

When the IP module receives an incoming datagram from local network module, it will pass the information to the user, if the addressed user had a pending RECV call, by a pseudo interrupt or similar mechanism; or it will notify the addressed user. If the user does not exists, an ICMP error message will be returned to the sender, and the datagram is discarded.

Table 7.1 Examples of Upper Layer Protocols Using IP

Decimal	Keyword	Protocol
1	ICMP	Internet Control Message
2	IGMP	Internet Group Management
3	GGP	Gateway-to-Gateway
6	TCP	Transmission Control
8	EGP	Exterior Gateway Protocol
9	IGP	private interior gateway
11	NVP-II	Network Voice Protocol
17	UDP	User Datagram
18	MUX	Multiplexing
20	HMP	Host Monitoring
21	PRM	Packet Radio Measurement
27	RDP	Reliable Data Protocol
28	IRTP	Internet Reliable Transaction
29	ISO-TP4	ISO Transport Protocol Class 4
30	NETBLT	Bulk Data Transfer Protocol
31	MFE-NSP	MFE Network Services Protocol
32	MERIT-INP	MERIT Internodal Protocol
33	SEP	Sequential Exchange Protocol
63		any local network

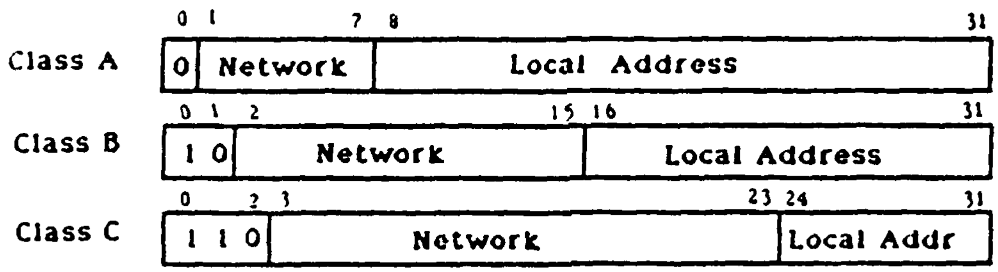
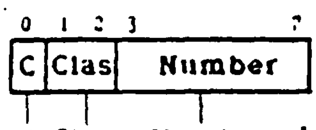


Figure 7.4 **The Classes of Internet Addresses**



Copied	Class	Number	Length	Description
	0			Control
	0	0	--	End of option list
	0	1	--	No operation
	0	2	11	Security
	0	3	var	Loose source routing
	0	7	var	Record route
	0	8	4	Stream ID
	0	9	var	Strick source routing
	2			Debugging & measurement

Figure 7.5 **Options of the IP**

8. INTERNET CONTROL MESSAGE PROTOCOL (ICMP)

The Internet Control Message Protocol (ICMP) is actually realized in many implementations as an integral part of the IP module. ICMP provides the destination hosts or gateways with the error reporting facilities about problems in communication environment. The ICMP messages are treated by the IP module as the data portion of the datagram, when the "protocol" field of the IP header equals "1". The format for most ICMP messages are described in Figure 8.1. And the types of the ICMP messages with different code are listed in Table 8.1.

9. USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) provides a transaction oriented procedure for application programs to send messages with minimum protocol overhead. It assumes the IP as underlying protocol, when protocol field value equals 17. In UDP, delivery and duplicate protection are not guaranteed. It is quite similar to the ISO TP-0 which assumes that the underlying Network Layer can provide reliable datagram services. It is more suitable to applications which requires the least protocol overhead, and it is not so critical to lose a small part of the data. The UDP header format is shown in Figure 9.1. The major uses of UDP is the Internet Name Server, and the Trivial File Transfer.

10. TRANSMISSION CONTROL PROTOCOL (TCP)

The Transmission Control Protocol (TCP) is a connection-oriented, highly reliable end-to-end protocol between hosts in the interconnected packet-switching computer networks. By connection-oriented, it means that the TCP establishes, and maintains the virtual circuit, or connection, between two communicating processes on the source and destination hosts. It is assumed that only unreliable datagram services are provided as the underlying Network Layer protocol, and the TCP must recover from data segments that are damaged, lost, duplicated, or delivered out of order. The TCP is located at the Host Level in the DDN internetworking architecture as described in Figure 6.2, and at the Transport and part of Session Layers in the ISO - OSI reference model. Following functions are provided by the TCP.

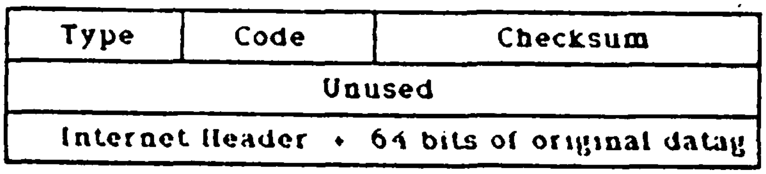


Figure 8.1 **ICMP Message Format**

Table 8.1 Some Examples of ICMP Message Type

Type	Code	Description
0	0	Echo Reply
3		Destination Unreachable
3	0	net unreachable
3	1	host unreachable
3	2	protocol unreachable
3	3	port unreachable
3	4	fragmentation needed and DF set
3	5	source route failed
4		Source Quench
5		Redirect
5	0	redirect for network
5	1	redirect for host
5	2	redirect for type of service & network
5	3	redirect for type of service & host
8	0	Echo
11		Time Exceeded
11	0	time to live exceeded in transit
11	1	fragment reassembly time exceeded
12	0	Parameter Problem
13	0	Timestamp
14	0	Timestamp Reply
15	0	Information Request
16	0	Information Reply

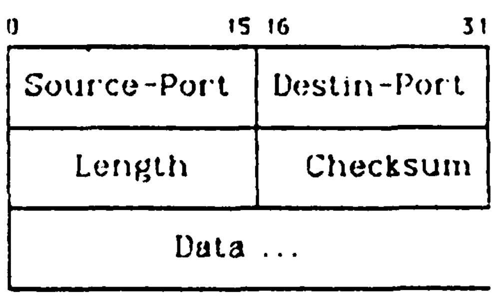


Figure 9.1 **User Datagram Protocol Header Format**

a. **Basic Data Transfer.** The TCP module accepts user data as a continuous stream of bytes, packages them appropriately into data segments with the TCP header, and calls the lower Network Layer module (e.g. the IP) to transfer the data. The user can also "push" the TCP module to deliver the accumulated data to the receiver immediately.

b. **Reliability.** Each byte in the data stream conceptually has a sequence number. The sequence number of the first byte in the data segment is indicated in the TCP header. If the sender does not receive positive acknowledgment in a certain timeout period from the receiver, the data segment is assumed damaged or lost, and retransmitted. A checksum is also added to each data segment for receiver to detect and discard the damaged data. The sequence number is also useful for receiver to detect and discard the duplicated data, and to relocate the data segments which are delivered out of order.

c. **Flow control.** The TCP provides the receiver with the "window" facility to indicate the allowed number of bytes that the sender may transmit before receiving further permission. Flow control operations proceed from this field.

d. **Multiplexing.** Different applications on hosts are assigned with different "port" addresses, so that many processes in a single host can use the same TCP communication facilities simultaneously. The internet and the port addresses make a socket which uniquely identifies one side of a connection.

e. **Connection.** The TCP initializes and maintains certain status information for each data stream for reliable data communications. It is called a connection, including sockets, sequence numbers, and window sizes.

f. **precedence and security.** The TCP makes use of the "type of service" field and the security option for the service.

10.1 OPERATIONAL MODEL DESCRIPTIONS

The TCP module is usually implemented as a device in the file system of an operating system. After establishing a connection (similar to OPEN device), the application processes in computer hosts transmit data by calling on the TCP module through device driver and passing buffers of data as arguments. The TCP packages the data from these buffers into segments with the control information to ensure the reliable transmission, and and calls on the IP module to transmit each segments to the destination TCP. The IP module will route the datagrams through local networks and intermediate gateways, fragment and reassemble them if necessary, as described in previous sections. The receiving TCP module will place the data from a segment into receiving user's buffer and notifies the user. The state transition of a connection in TCP over which data are transferred can be described by a finite state machine, as shown in Figure 10.1.

The **CLOSED** state represents no connection. If the local TCP receives a passive OPEN call from TCP user, it will create a transmission control block (TCB), fill the TCB with control parameters from the OPEN call, and change its state to **LISTEN** to incoming OPEN calls. On an active OPEN call, the TCB will be created, and the TCP will start the procedure to establish the connection at once.

TCP uses three-way handshake for the connection establishment. It is necessary because the initial sequence numbers are selected using local clock in a 4.55 hour period, not a global clock. The **LISTEN** TCP has no way of knowing whether it is an old delayed OPEN or not, and it must ask OPEN initiator to verify it.

The data transfer proceeds in the **ESTAB** state.

When the local TCP receives a CLOSE call from user, it will send a FIN packet to remote TCP, and change to **FIN WAIT-1** state. The remote TCP will send back an ACK, notify its user through the return parameters of any user call, and stay in **CLOSE WAIT**. Data transfer can still be performed at this moment. When the remote TCP receives a CLOSE from its user, it will send a FIN packet, wait for acknowledgment, and change to **CLOSED** state. The local TCP will send an ACK packet to remote TCP, wait for 2 MSec **TIME-WAIT**, and change to **CLOSED** state.

The state of the connection, along with other control parameters, are stored in the transmission control block (TCB). Those parameters include send sequence variables (initial send sequence number, segment sequence number for last window update and send window, send unacknowledged, next send sequence number, etc.); receive sequence variables (similar to send sequence variables); and current segment variables (segment sequence number, acknowledgment number, length, window, urgent pointer, and precedence value). TCP module maintains the connection status upon incoming user calls or incoming packets.

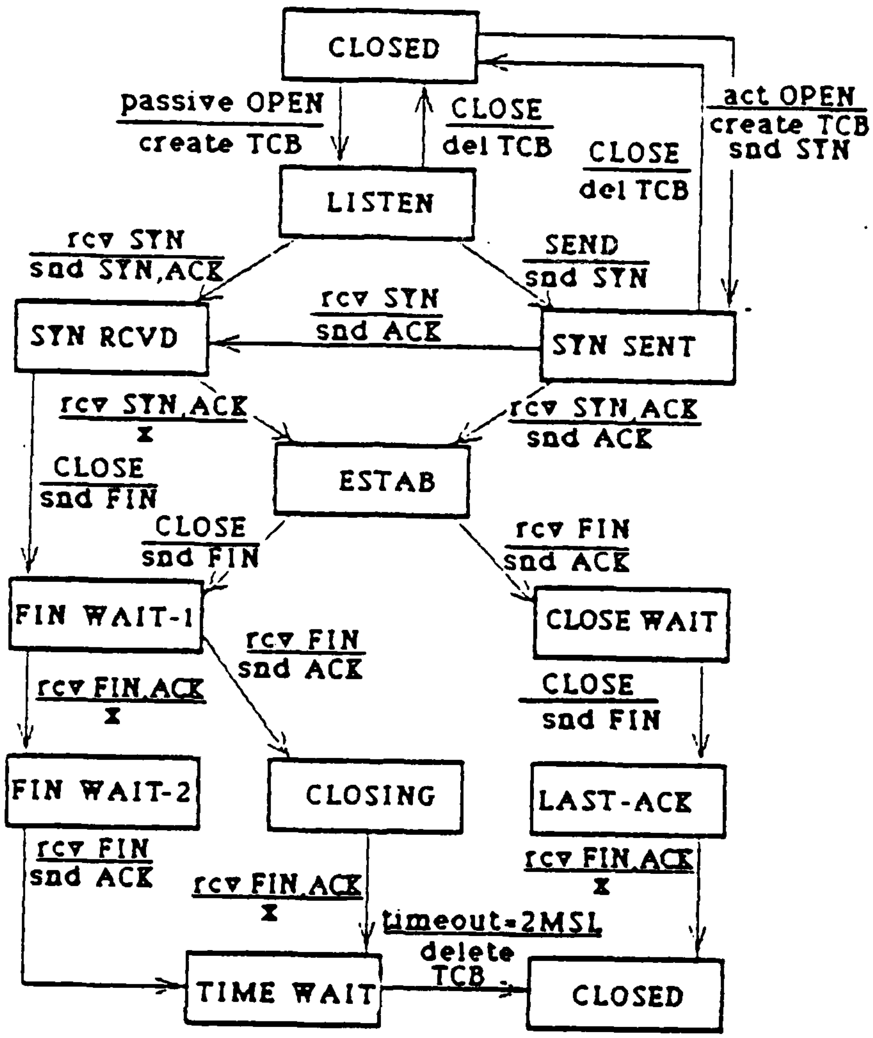


Figure 10.1 TCP Connection State Diagram

10.2 PROTOCOL SPECIFICATIONS

The TCP header, described in Figure 10.2, is sent along with user data as the data portion of the internet datagram, immediately following the IP header.

Source Port (16 bits)

Destination Port (16 bits)

Ports are used in the TCP to name the ends of connections for the communication. For the purpose of providing services to unknown callers, some service contact ports, also called "well-known ports", are pre-defined, and some examples are listed below in Table 10.1.

Sequence Number (32 bits): The sequence number of the first byte in the data segment. If the packet is SYN for OPEN request, this field contains the Initial Sequence Number (ISN).

Acknowledgment Number (32 bits): The sequence number of the data byte the receiver is expecting to receive, when the ACK control bit is set.

Data Offset (4 bits): The length of the TCP header in 32 bits word.

Control Bits (6 bits): The Bit pattern is shown in Figure 10.3, and the meaning is explained below:

URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Finish, no more data from sender

Window (16 bits): The number of bytes the receiver is will to accept. It is started from the byte indicated in the acknowledgment field.

Checksum (16 bits): The checksum for all 16-bit words in the header text, and a 96-bit pseudo header. It is calculated as the one's complement of the one's complement sum of all 16-bit words.

Urgent Pointer (16 bits): The offset of the end of urgent data from the sequence number, when the URG control bit is set.

Options (variable length): TCP module must implement all options.

Kind	Length	Meaning
0	-	end of option list
1	-	no-op
2	4	maximum segment size, indicated in 16-bit field

10.3 THE INTERFACE DESCRIPTIONS

There exist two kinds of interfaces: the User/TCP interface, and the TCP/IP interface. The upper protocol layers use the user/TCP interface to communicate with TCP.

10.3.1 The User/TCP Interface

The User/TCP interface is realized by the calls by the TCP users to the TCP module.

OPEN (local_port, foreign_socket, active/passive, [,timeout] [,precedence] [,security/compartment] [,options])
==> local_conn_name

This call is used by TCP user to establish a connection with the destination host with the address indicated by the foreign_socket.

If the active/passive flag is set to passive, it is a call to LISTEN for incoming connection, for any call if the foreign_socket is unspecified, or for a particular call if the foreign_socket is fully specified.

The timeout gives a time limit on delivering the user data. If the user data is not successfully delivered within the time limit, the connection is aborted. The default value for timeout is 5 minutes.

The TCP module accept incoming requests only if the security/compartment information is exactly the same, and the precedence is equal or higher than that in the OPEN call.

SEND (local_conn_name, buf_addr, bytes, PUSH, URGENT, [,timeout])

This call is used by the TCP user to send data over the TCP connection.

In handling the data transfer, the TCP module will send the data in the buffer immediately to the receiver if the PUSH flag is set; otherwise, the data may be combined with data from subsequent SENDs for the efficiency.

When the URGENT flag and the urgent pointer is set, the receiving TCP will signal the TCP user if the data preceding the urgent pointer has not been consumed yet. It is used to stimulate the receiver to process the urgent data.

RECEIVE (local_conn_name, buffer, bytes) ==> bytes, URGENT, PUSH

This call is used by the TCP user to allocate a receiving buffer associated with the connection. The buffer will be filled with as much incoming data as it can hold if there is no PUSH flag set in incoming packet. Otherwise, the buffer will be returned with partially filled data.

CLOSE (local_conn_name)

This call is used to close the specified connection, after gracefully transmitting outstanding SENDS. In this case, the CLOSE from TCP user means only "no more data to SEND", but not means "no data will be received".

ABORT (local_conn_name)

This call is used to cause all pending SENDs and RECEIVEs to be aborted, the TCB to be removed, and special RESET message to be sent to the TCP module on the other side.

STATUS (local_conn_name) ==> status data

The status data returned includes: local and foreign sockets, local connection name, receive and send window, connection state, number of buffers awaiting acknowledgment and pending receipt, urgent state, precedence, security/compartment, and transmission timeout.

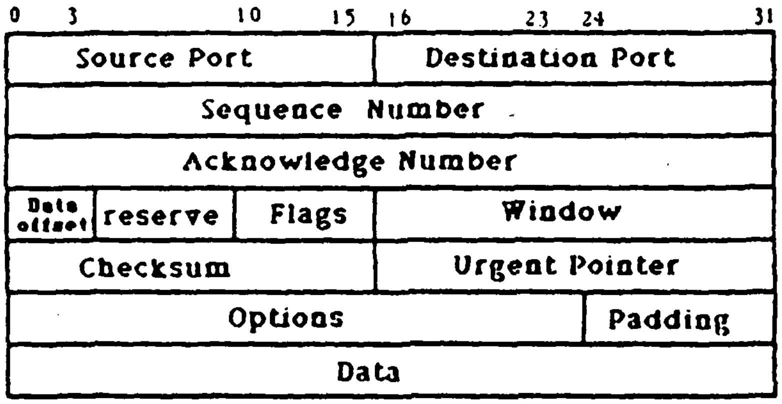


Figure 10.2 **The TCP Header Format**

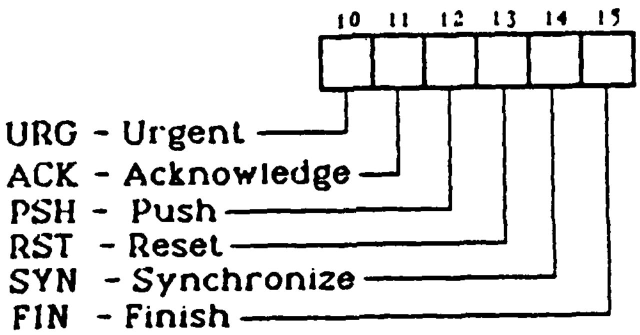


Figure 10.3 **The Bit Pattern of Control Bits**

Table 10.1 **Examples of TCP Ports**

Decimal	Keyword	Description
5	RJE	Remote Job Entry
7	ECHO	Echo
9	DISCARD	Discard
11	USERS	Active Users
13	DAYTIME	Daytime
19	CHARGEN	Character Generator
20	FTP-DATA	File Transfer [Data]
21	FTP	File Transfer [Control]
23	TELNET	Telnet
25	SMTP	Simple Mail Transfer
37	TIME	Time
39	RLP	Resource Location Protocol
41	GRAPHICS	Graphics
42	NAMESERVER	Host Name Server
43	NICNAME	Who Is
49	LOGIN	Login Host Protocol
51	LA-MAINT	IMP Logical Address Maintenance
53	DOMAIN	Domain Name Server
63	VIA-FTP	VIA Systems - FTP
65	TACACS-DS	TACACS-Database Service
67	BOOTPS	Bootstrap Protocol Server
68	BOOTPC	Bootstrap Protocol Client
69	TFTP	Trivial File Transfer
71	NETRJS-1	Remote Job Service
72	NETRJS-2	Remote Job Service
73	NETRJS-3	Remote Job Service
74	NETRJS-4	Remote Job Service
93	DCP	Device Control Protocol
95	SUPDUP	SUPDUP
97	SWIFT-RVF	Swift Remote Vitural File Protocol
101	HOSTNAME	NIC Host Name Server
102	ISO-TSAP	ISO-TSAP
103	X400	X400
104	X400-SND	X400-SND
105	CSNET-NS	Mailbox Name Nameserver
107	RTELNET	Remote Telnet Service
109	POP-2	Post Office Protocol
113	AUTH	Authentication Service
115	SFTP	Simple File Transfer Protocol
117	UUCP-PATH	UUCP Path Service
119	NNTP	Network News Transfer Protocol
123	NTP	Network Time Protocol
125	LOCUS-MAP	Locus PC-Interface Net Map Server
127	LOCUS-CON	Locus PC-Interface Conn Server
129	PWDGEN	Password Generator Protocol
133	STATSRV	Statistics Service
136	PROFILE	PROFILE Naming System
137	NETBIOS-NS	NETBIOS Name Service
138	NETBIOS-DGM	NETBIOS Datagram Service
139	NETBIOS-SSN	NETBIOS Session Service
243	SUR-MEAS	Survey Measurement

10.3.2 The TCP/IP interface

The TCP module calls IP module to Send and Receive data over networks. The interface to IP is described in Section 7.3. The default values for the type of service in IP are listed below:

Precedence:	routine
Delay:	normal
Throughput:	normal
reliability:	normal
Time to Live:	1 minute

PART III. THE ISO/OSI PROTOCOL ARCHITECTURE

In this and following chapters, the OSI basic reference model and the ISO Session, Transport, and Network Layer protocols are reviewed. This section of ISO protocol studies and previous section of TCP/IP protocol studies should prepare the reader for the transport layer gateway discussion in the next section.

11. THE ISO AND THE OSI REFERENCE MODEL

To avoid the confusion with the ISO standard's documentation, the ISO organization and the standards documentation procedures are explained in this chapter. It also describes the OSI basic reference model, and gives the global view of the ISO protocols.

11.1 THE ISO AND THE ISO STANDARDS DOCUMENTING

The International Standards Organization, or ISO, is a world-wide organization for producing the *international* standards in all areas. It is divided into Technical Committees (TC), SubCommittees (SC), and Working Groups (WG) to deal with problems in specific branches of different areas. The technical committee 97 (TC97) is specialized in the information processing systems. There are three subcommittees in TC97 related to the Open System Interconnection (OSI), as described in Figure 11.1. Other working groups not in the figure include Computer Graphics, Conceptual Schema, Database Languages, Data Descriptive File, and Text and Office Systems.

All international standards are processed through the ISO in a similar way. Each working group works on the Working Drafts for the required standards, labeled as

ISO/TC#/SC#/ [WG#/] N#

After step-by-step approval, it becomes Draft Proposals (DP), Draft International Standards (DIS). After the approval on a DIS ballot, the DIS becomes an international standard (IS). DP, DIS, and IS are labeled by

ISO DP #, or
ISO DIS #, or
IS #

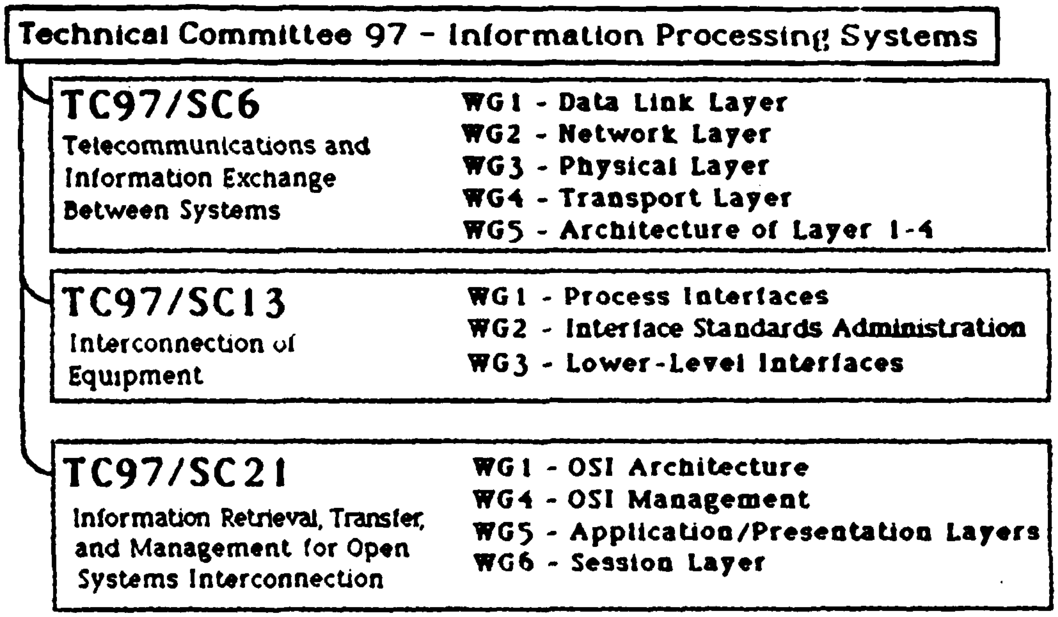


Figure 11.1 ISO Organization Chart

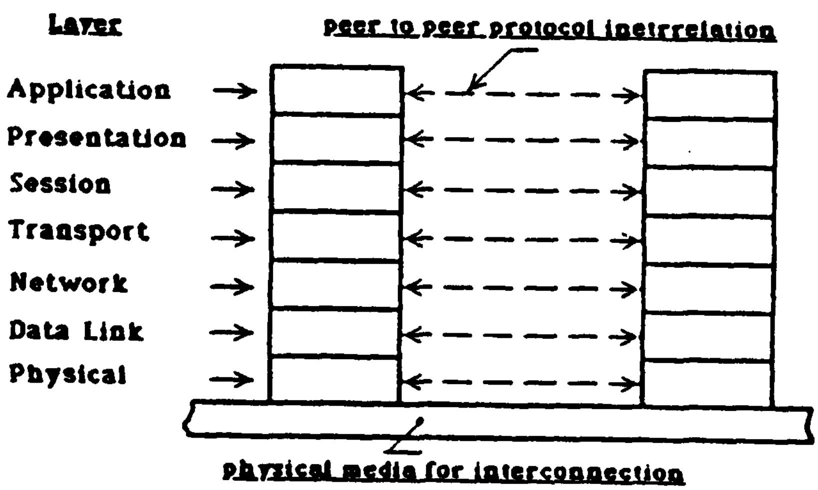


Figure 11.2 OSI Basic Reference Model

where the number # is kept the same. The amendment to the international standards is produced in the same way, except labeled as Working Draft, Draft Proposed Addendum (DPAD #), Draft Addendum (DAD #), and Addendum (AD#). The ISO standards are reviewed periodically, usually on a five-year cycle.

11.2 THE OSI BASIC REFERENCE MODEL

The Open System Interconnection (OSI) basic reference model was developed to provide a common basis for the coordination of standards development while allowing existing standards to be placed into perspective within the model. The model provides a conceptual and functional framework for the definition of services and protocols within the boundaries It includes sufficient flexibility to accommodate advances in technology and expansion in user demands, so as to allow teams to work productively and independently in standards development. It is intended to ease the tasks to identify areas for developing or improving standards, and to provide a common reference for maintaining consistency of all related standards.

In ISO standard 7498 [ISO 7498], the definitions and environment of Open system Interconnection, and the modeling of the OSI environment is introduced. It describes the concept of a layered architecture as a basic structuring technique. Open systems in the OSI reference model are decomposed into seven layers, as shown in Figure 11.2. Similar functions in OSI are grouped into the same layer so as to localize changes, to minimize and standardize interfaces, and to facilitate creating sublayers for optional, specific functions. This standard also describes, by layers, the definition, the services provided to upper layer, and the functions within the layer for each of the seven layers in the OSI reference model. Table 11.1 summarizes the OSI layers and services.

11.3 ISO STANDARDS

The ISO standards documents currently available cover a large range of different areas, and many of them are still the draft proposals (DP). Table 11.2 contains a list of ISO documents related to the OSI.

Table 11.1 OSI Layers and Services

Layer	Services
7 Application Layer	•provides end-users with comprehensive interface into all kinds of distributed information services (document distribution, electronic mail, distributed transaction processing)
6 Presentation Layer	•identifies, negotiates communications transfer syntax •formats data (binary, ASCII, EBCDIC, graphics, numerics) •provides special transformation (compression, encryption)
5 Session Layer	•provides session-connection establishment, and release •supports application process dialog •exchanges normal and expedited data
4 Transport Layer	•provides for establishment, data transfer, and termination of logical connections between session entities •provides end-to-end information interchange and control •provides error detection and recovery
3 Network Layer	•provides network routing and connection services •segments and blocks network messages •provides expedited data transfer •provides error detection, recovery, and notification
2 Data Link Layer	•initializes and disconnect data link between adjacent nodes •transfers data over link •provides error detection and correction
1 Physical Layer	•provides physical interface through electrical, mechanical, procedural, and functional means

Table 11.2 ISO / OSI Documents

ISO/OSI Doc	Description
General:	
IS 7498	OSI Basic Reference Model
TR 8509	Service Conventions
DP 8807	LOTOS - Description of the Temporal Ordering Specification Language
DP 9074	ESTELLE - A Formal Description Based on an Extended State Transition Model
Application Layer:	
DIS 8571	File Transfer, Access and Management
DIS 8649	Definition of Common Application Service Elements
DIS 8650	Specification of Protocol for Common Application Service Elements
DIS 9040	Basic Class Virtual Terminal Service
DIS 9041	Basic Class Virtual Terminal Protocol
Presentation Layer:	
DIS 8822	Connection Oriented Presentation Service Definition
DIS 8823	Connection Oriented Presentation Protocol Definition
DIS 8824	Specification of Abstract Syntax Notation One (ASN.1)
DIS 8825	Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)
Session Layer:	
DIS 8326	Basic Connection-Oriented Session Service Definition
DIS 8327	Basic Connection-Oriented Session protocol Specification
Transport Layer:	
IS 8072	Transport Service Definition
IS 8073	Transport Protocol Specification
DIS 8602	Protocol for Providing the Connectionless-mode Transport Service

Network Layer:

DIS 8208	X.25 Packet Level Protocol for Data Terminal Equipment
DIS 8348	Network Service Definition
IS 8473	Protocol for Providing the Connectionless-mode Network Service
DIS 8648	Internal Organization of the Network Layer
DIS 8878	Use of X.25 to Provide the OSI Connection-mode Network Service
DIS 8881	Use of the X.25 Packet Level Protocol in Local Area Networks
DIS 8882	X.25--DTE Conformance Testing

Data Link Layer:

IS 4335	HDLC Procedures - Consolidation of Elements of Procedures
DIS 7478	Multilink Procedures
IS 7776	HDLC Procedures - X.25 LAPB-Compatible DTE Data Link Procedures
IS 7809	HDLC Procedures - Consolidation of Classes of Procedures
IS 8471	HDLC Balanced Classes of Procedures - Data Link Layer Address Resolution/Negotiation in Switched Environments
DIS 8802	Local Area Networks
DIS 8885	HDLC Procedures - General Purpose XID Frame Information Field Content and Format
DIS 8886	Data Link Service Definition for OSI

Physical Layer:

SC6/N3631	Physical Service Definition
DIS 2110	25-pin DTE/DCE Interface Connector and Pin Assignments
DIS 4902	37-pin DTE/DCE Interface Connector and Pin Assignments
DIS 4903	15-pin DTE/DCE Interface Connector and Pin Assignments
DIS 8480	DTE/DCE Interface Back-up Control Operation Using the 25 Pin Connector
IS 8481	DTE to DTE Physical Connection Using X.24 Interchange Circuits with DTE Provided Timing
DIS 8482	Twisted Pair Multipoint Interconnections
DIS 8877	Interface Connector and Contact Assignments for ISDN Basic Access Interface Located at Reference Points S and T
DIS 9067	Automatic Fault Isolation Procedures Using Test Loops

11.4 CONCEPTS OF CONNECTION- AND CONNECTIONLESS-MODE

Another important concept related to this protocol conversion task is the concepts of Connection-mode and Connectionless-mode transmission. In the ISO documents, these modes are called connection-oriented and connectionless services, respectively.

A connection, in the formal terminology of the OSI Reference Model, is an association between two or more peer-entities established for data transfer. In addition to three distinct phases of Connection Establishment, Data Transfer, and Connection Release operations, and distinguishable lifetime of those phases, it has following fundamental characteristics:

1) It involves establishing and maintaining of two or more party agreement concerning data transfer between the peer-entities and the layer providing the service;

2) It allows the negotiation among all the parties concerned of the parameters and options that will govern the data transfer;

3) It provides connection id in which the overhead associated with address resolution and transmission can be avoided during the data transfer phase;

4) It provides a context in which successive data units transferred between the peer-entities are logically related, and therefore with the preservation of sequence and provision of flow control.

These connection oriented characteristics are attractive in a wide range of applications which call for relatively long lived, stream-oriented interactions between entities in stable configurations.

In contrast, the connectionless-mode transmission, or so called "message-mode", "datagram", "transaction mode", and "connection free", has always played an important role. It is basically the transmission of a single data unit from a source service-access-point to one or more service-access-points without establishing a connection, by performing a single service access. It has following fundamental characteristics:

1) It requires only a pre-existing association between the peer-entities involved, without any peer-to-peer agreement in using the service;

2) All the information required to deliver a data unit

(addresses, quality of service, options, etc.), together with data transmitted, is presented to the layer providing the connectionless-mode service in any single service access. As a result, it may also true that

3) Each data unit transmitted is entirely self-contained and can be routed independently;

4) Copies of a data unit may be transmitted to a number of destination addresses.

The characteristics of the connectionless-mode transmission are attractive in applications which involve short-term request/response interactions, exhibit a high level redundancy, must be flexibly reconfigurable, or derive no benefit from guaranteed in-sequence data delivery.

Protocols in some layers are described separately in the transmission modes. DIS 8473 defines the standard for providing connectionless-mode network service (CLNS), very similar to that of DDN Internet Protocol (IP). Both DDN TCP and ISO-TP feature connection-mode transport service. The next two sections describe the ISO Network and Transport layer services and protocols.

12. THE NETWORK LAYER

The Network Layer provides the means to establish, maintain and terminate network-connections, and the functional and procedural means to exchange network-service-data-units between transport-entities, with the independence from routing and relay considerations.

12.1 NETWORK SERVICE DEFINITION (DIS 8348)

This standard defines the Network Service in OSI Reference Model, and Network Service primitives, mainly for connection-mode type.

12.1.1 The Network Service

The Network Service provides for the transparent data transfer between its users.

1) End-to-end data transfer:

All routing and relaying functions are performed by the Network Service provider.

Independent of underlying transmission media. It relieves the users from all concerns regarding how data are transferred over various heterogeneous subnetworks.

Data transparency. Data transferred are not restricted for the content, format, coding, structure, or meaning.

2) User addressing. It utilizes a system of addressing (NSAP addressing) to allow users to refer unambiguously to one another.

3) Quality of Service Selection. It provides the users with a means to request and to agree the quality of service, such as

throughput;
transit delay;
transfer failure probability; and
residual error rate which is defined as

$$RER = \frac{N(lost) + N(error) - N(extra)}{N(total)}$$

where N is the number of packets.

More services are included in the connection-mode network service:

1) Network-connection establishment and release;

2) Reset to predefined state;

3) Flow control;

4) Expedited data transfer;

5) More quality of service selection:

Network-connection establishment delay;
Network-connection establishment failure probability;
Network-connection resilience;
Network-connection release delay;
Network-connection release failure probability;
Network-connection protection;
Network-connection priority;
Maximum acceptable cost.

12.1.2 Network Service Primitives

Table 12.1 gives a summary of connection-mode Network Service primitives.

12.2 NETWORK PROTOCOL SPECIFICATION (IS 8473: CONNECTIONLESS-MODE)

The protocol specified in this standard provides the connectionless-mode Network Service, very similar to the internetwork datagram service provided in the DDN Internet Protocol (IP).

12.2.1 Underlying Service and Functions

Table 12.2 shows the Connectionless-mode Network Service primitives provided by this protocol. The underlying connectionless-mode service, very similar to the service primitive in Table 12.2, may be obtained either directly from a connectionless-mode real subnetwork in the way specified in this standard, or indirectly through the Subnetwork Dependent Convergence Function (SNDCF) or Protocol (SNDCP) over a connection-mode real subnetwork, as described in DIS 8648, Internal Organization of the Network Layer.

The types of functions provided by this protocol are summarized in Table 12.3. Type 1 functions are required for all implementation. Type 2 and 3 functions are optional. If Type 2 function is selected in a PDU but not implemented locally, then the PDU is discarded, and an Error Report PDU is generated and forwarded to the originating network-entity. If Type 3 function is selected but not locally implemented, it is processed just like not selected.

12.2.2 PDU Structure Specification

The Protocol Data Unit (PDU) structure specified in this standard is shown in Figure 12.1. It is divided into four parts: the fixed part, the address part, the segmentation part, and the options part. The data portion follows the options part.

Table 12.1 IS 8348: Connection-mode Network Service Primitives

Phase	Service	Primitive
NC Establishment	NC Establishment	N-CONNECT request N-CONNECT indication N-CONNECT response N-CONNECT confirm
Data Transfer	Data Transfer	N-DATA request N-DATA indication
	Receipt-Confirmation*	N-DATA-ACKNOWLEDGE request N-DATA-ACKNOWLEDGE indication
	Expedited data transfer *	N-EXPEDITED-DATA request N-EXPEDITED-DATA indication
	Reset	N-RESET request N-RESET indication N-RESET response N-RESET confirm
NC Release	NC Release	N-DISCONNECT request N-DISCONNECT indication

* Optional

Table 12.2 DIS 8473: Connectionless-mode Network Service Primitives

Primitive	Parameters
N-UNITDATA.request N-UNITDATA.indication	NS-source-address, NS-desti-address, NS-quality-of-service, NS-userdata

Table 12.3 DIS 8473: Connectionless-mode Network Service Functions

Type-1:

PDU composition and decomposition;
Header format analysis and PDU header error detection;
PDU lifetime control;
Route and forward PDU;
Segmentation and reassembly;
Discard PDU;
Error reporting

Type-2:

Security;
Complete source routing;
Complete route recording

Type-3:

Partial record routing;
Partial route recording;
Priority;
QoS maintenance;
Congestion notification;
Padding

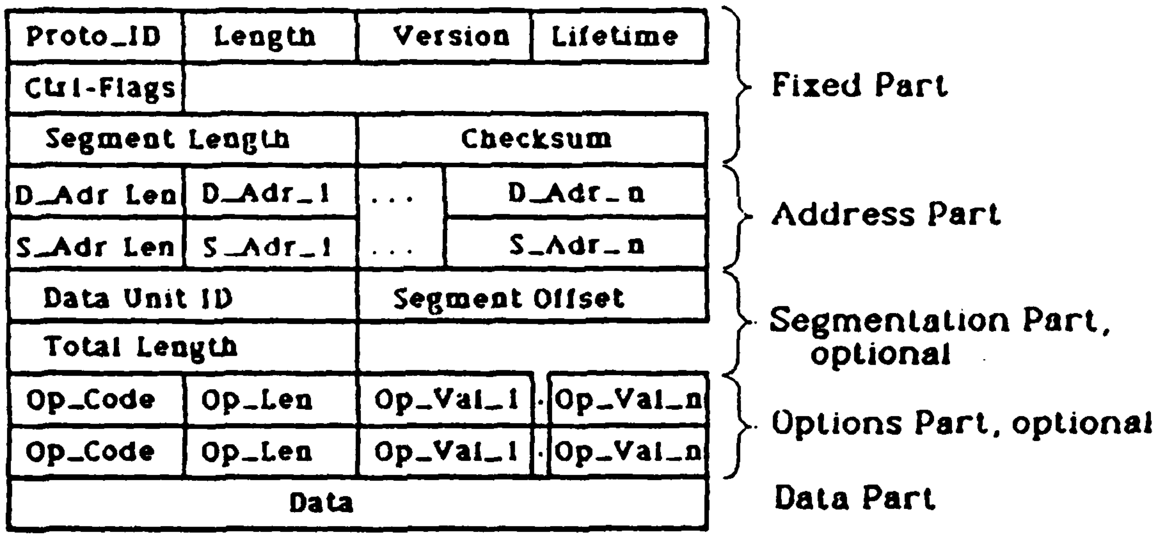


Figure 12.1 DIS 8473: PDU Structure

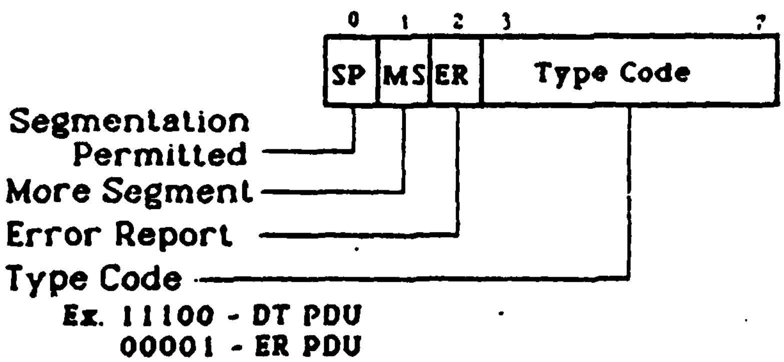


Figure 12.2 DIS 8473: Control Flags

Network Layer Protocol Identifier (1 Octet):

1000 0001: Network Layer protocol as ISO 8473
0000 0000: Inactive Network Layer protocol subset

Length Indicator (1 Octet): The length in octets of the header. The maximum is 254.

Version (1 octet): Value 0000 0001 is for this version of the standard.

PDU Lifetime (1 Octet): The remaining lifetime of the PDU, in units of 500 ms.

Flags (1 Octet): The flag bits are defined in Figure 12.2.

PDU Segment Length (2 Octets): The entire length of the PDU in octets.

Checksum (2 Octet): It is computed on the entire header.

Destination-Address Length (1 Octet)
Source-Address Length (1 Octet): The length of following address field in octets. These values may vary according to the application.

Destination/Source Address: These are the addresses defined in ISO 8348/AD2. The lengths of the field are variable, as indicated in the "Address Length" fields.

Data Unit Identifier (2 Octets): Identifies the Initial PDU (un-segmented) and all the Derived PDU contain the same value.

Segment Offset (2 Octets): For each Derived PDU, it specifies the relative position of the data segment with respect to the start of data part of Initial PDU.

Total Length (2 Octets): Specifies the total length of the Initial PDU in octets, including both the header and data.

For each option, it includes option-code, option-length, and option-value three fields.

Option Code (1 Octet): It specifies different options. Some example are shown in Table 12.4.

Table 12.4 Examples of the Options in DIS 8473

Option-Code	Length	Value	Description
1100 1100	var	any	padding
1100 0011	1	*	quality of service
1100 0101	var	*	security
1100 1000	var	*	source routing
1100 1011	var	*	record route
1100 1100	1	0-0F	priority

Option Length (1 Octet): Specifies the length of option value in octets.

Option Value: Contains the value of the option with the length as indicated in the option-length field.

Besides, the provision of the underlying service from the subnetwork is also described in this standard.

13. THE TRANSPORT LAYER

The Transport Layer in OSI Reference Model provides transparent data transfer between its users, and relieves them from any concerns with the details for the reliable and cost effective data transfer. It optimizes the use of the available network service to provide required performance at minimum cost. The protocol defined at this layer have end-to-end significance, between correspondent transport-entities. Therefore the transport layer in OSI is end system oriented, and the transport protocols operate only between OSI end systems.

13.1 TRANSPORT SERVICE DEFINITION (IS 8072)

This standard defines the Transport Services in the OSI Reference Model, and the Transport Service primitives with associated state transition.

13.1.1 Transport Service (TS)

The transport service provides for the transparent, reliable data transfer between session entities. Transport services are:

1) Transport-Connection (TC) establishment and release in a synchronized manner:

   The Transport Service utilizes an addressing system (TSAP) to allow users to refer unambiguously to one another.

   It has end-to-end significance.

   User can, at the TC establishment time, request, negotiate and agree on a certain quality of service (QoS).

   Reset the TC to predefined state.

2) End-to-end reliable data transfer:

   Data transparency. Data transferred are not restricted for the content, format, coding, structure, or meaning.

   Flow control.

   Expedited data transfer.

3) Quality of Service Selection:

   TC establishment delay;
   TC establishment failure probability;
   throughput;
   transit delay;
   transfer failure probability; and
   residual error rate which is defined as

$$RER = \frac{N(lost) + N(error) - N(extra)}{N(total)}$$

   TC resilience;
   TC release delay;
   TC release failure probability;
   TC protection;
   TC priority.

13.1.2 Transport Service Primitives

Table 13.1 gives a list of Transport Service (TS) primitives in different phases. The allowed sequence of TS primitives with the state transition is shown in Figure 13.1.

13.2 TRANSPORT PROTOCOL SPECIFICATION (IS 8073)

This standard specifies the protocol for the connection oriented transport services.

13.2.1 Transport Layer Functions

Generally speaking, the transport layer should provide the TS users with following functions:

1) Assignment of the transport connections to network connections, either existing one or newly created one, with respect to:

   Resynchronization
   Reassignment after failure
   Splitting and recombining
   Multiplexing and demultiplexing

2) Connection establishment, with the respect to:

   Connection refusal

3) Transport Protocol Data Unit (TPDU) transfer, with respect to:

   Data TPDU numbering
   Resequencing
   Retention until acknowledgement of TPDUs
   Retransmission on time-out
   Expedited data transfer
   Segmenting and reassembling
   Concatenation and separation
   Association of received TPDUs with TC
   Treatment of protocol errors
   Explicit flow control
   Checksum

4) Normal release of TC, with the respect to:

   Frozen references

Table 13.1 IS 8072: Transport Service Primitives

Phase	Service	Primitive
TC Establishment	TC Establishment	T-CONNECT request T-CONNECT indication T-CONNECT response T-CONNECT confirm
Data Transfer	Data Transfer	T-DATA request T-DATA indication
	Expedited data transfer *	T-EXPEDITED-DATA request T-EXPEDITED-DATA indication
TC Release +	TC Release	T-DISCONNECT request T-DISCONNECT indication

* Optional

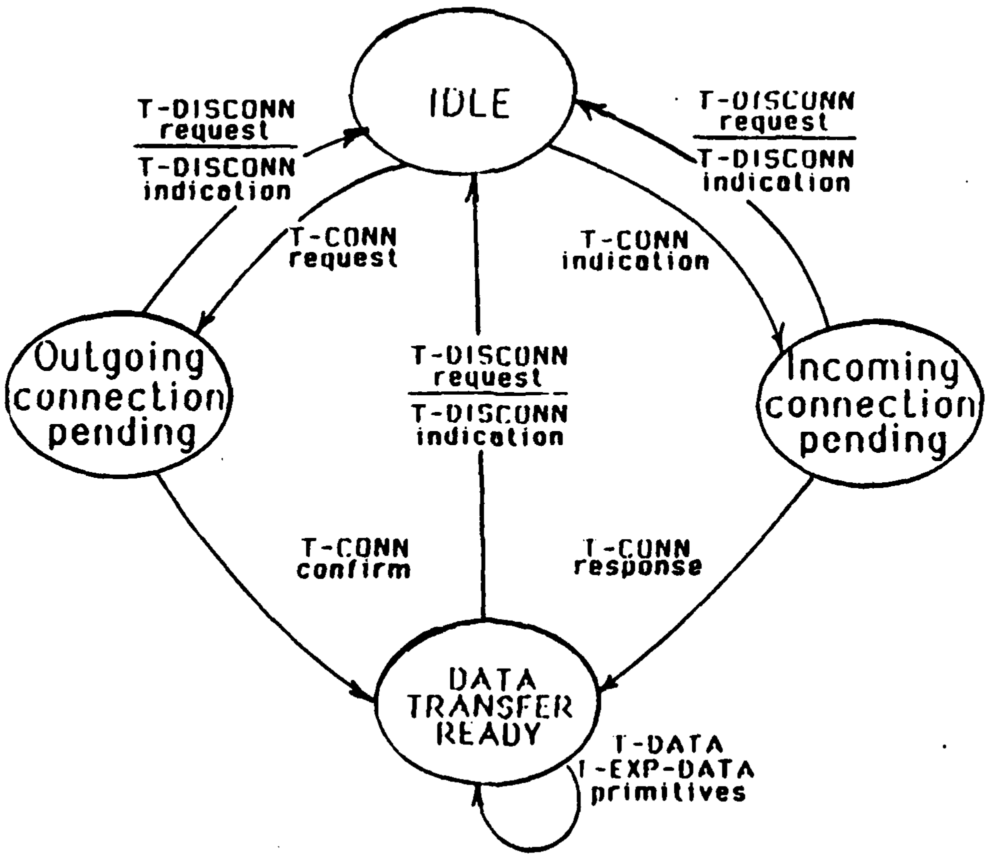


Figure 13.1 State Transition with Possible Allowed TS Primitives

5) Error release, with the respect to:

Inactivity control

13.2.2 Transport Protocol Classes

The IS 8073 standard defines five classes of transport connections with different level of functionalities. It is assumed to use three underlying choices of network connections:

Type A: with acceptable residual error rate and acceptable rate of signaled errors;

Type B: with acceptable residual error rate but unacceptable rate of signaled errors;

Type C: with unacceptable residual error rate.

Five classes of transport connections are specified based on on three classes of network connections:

Class 0: simple class
- the simplest type transport connection,
- compatible with CCITT T.70 for teletex terminal,
- use type A network connections;

Class 1: basic error recovery class
- basic transport connection with recovery from network disconnect or reset,
- use type B network connections;

Class 2: multiplexing class
- multiplexing several transport connections onto a single network connection, with/out explicit flow control,
- use type A network connections;

Class 3: error recovery and multiplexing class
- characteristics of class 2, plus recovery from network disconnect or reset,
- use type B network connections;

Class 4: error detection and recovery class
- characteristics of class 3, plus detection and recovery from errors of low grade service,
- use type C network connections.

The use of classes, as well as options for those functions within classes, is negotiated during connection establishment.

13.2.3 TPDU Structure Specification

13.2.3.1 TPDU General Structure

The structure of the Transport Protocol Data Units (TPDUs) can be divided into four parts, as shown in Figure 13.2.

Length Indicate (LI, 1 Octet): The length of the header in octets.

Fixed Part: Contains frequently occurring parameters, such TPDU code, etc. The format and the structure of this part is different for different TPDU.

Variable Part: Contains less frequently used information. It has the structure as described in Figure 13.3.

Following section will explain some of the TPDU structures and field definitions as examples.

13.2.3.2 Connection Request (CR) and Connection Confirm (CC)

The structures of the **Connection Request** (CR) and the **Connection Confirm** (CC) TPDUs are shown in Figure 13.4.

CDT -- Initial credit allocation, 0000 for Class 0,1

DST-REF -- Reference for the Destination transport entity, zero for CR TPDU

SRC-REF -- Reference for Source (initiating) transport entity

CLASS -- Bit 8-5 (left) defines the preferred class;
Bit 2 indicates the "Extended Format";
Bit 1 indicates no use of explicit flow control in class 2

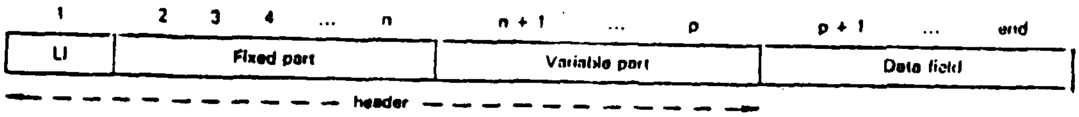


Figure 13.2 TPDU Structure

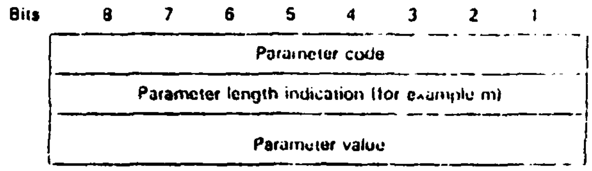


Figure 13.3 The Structure of TPDU Variable Part

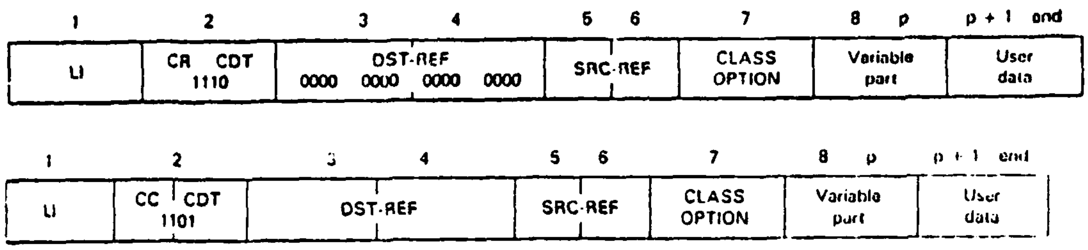


Figure 13.4 Structure of Connection Request(CR)/Confirm(CC) TPDUs

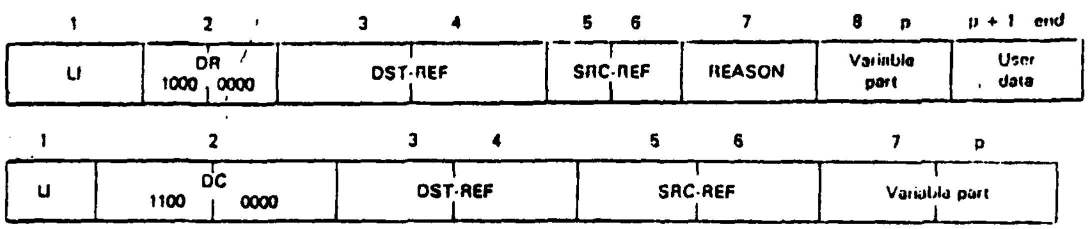


Figure 13.5 Structure of Disconnection Request(DR)/Confirm(DC) TPDUs

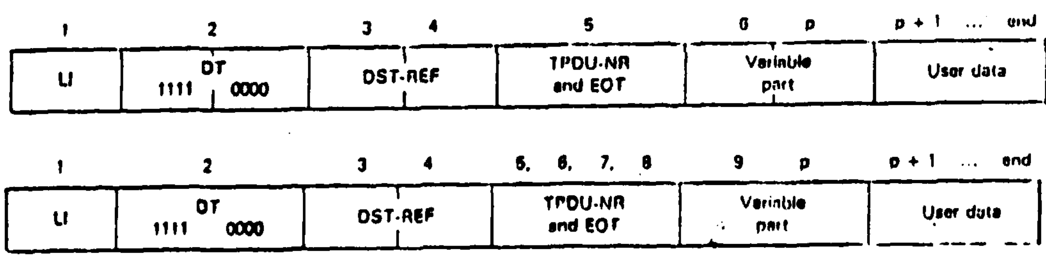


Figure 13.6 The Structure of Data (DT) TPDU

13.2.3.3 Disconnection Request (DR) and Disconnection Confirm (DC)

The structures of the **Disconnection Request** (DR) and **Disconnection Confirm** (DC) are shown in Figure 13.5.

REASON -- It defines the reason for disconnecting the transport connection

13.2.3.4 Data (DT)

The structure of **Data** (DT) TPDUs are shown in Figure 13.6.

TPDU-NR -- TPDU Sequence number

EOT -- The end of TPDU

The second structure shown in the figure has an extended format in **YR-TU-NR**.

13.2.3.5 Acknowledge (AK)

The structure of **Acknowledge** (AK) TPDU is shown in Figure 13.7.

YR-TU-NR -- The sequence number of next expected DT TPDU.

13.2.3.6 Expedited Data (ED) and Expedited Acknowledge (EA)

The structures of **Expedited Data** (ED) and **Expedited Acknowledge** (EA) are shown in Figure 13.8. ED and EA have very similar structure compared to the one of DT and AK.

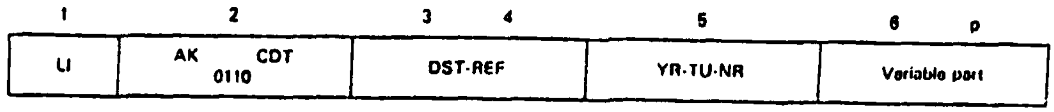


Figure 13.7 The Structure of Acknowledgement (AK) TPDU

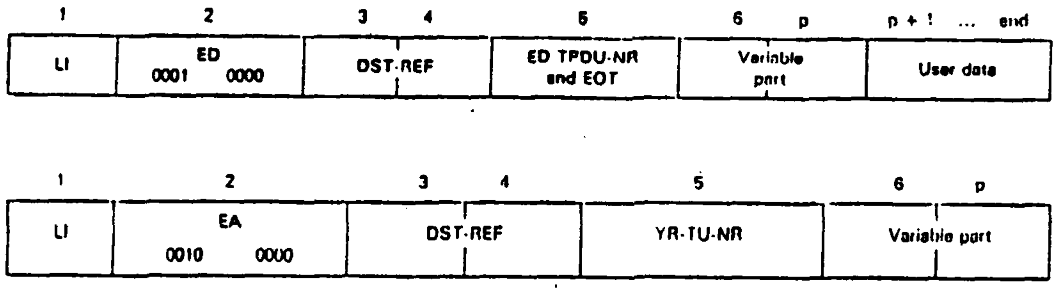


Figure 13.8 Structure of Expedited Data(ED)/Acknowledgement(EA) TPDUs

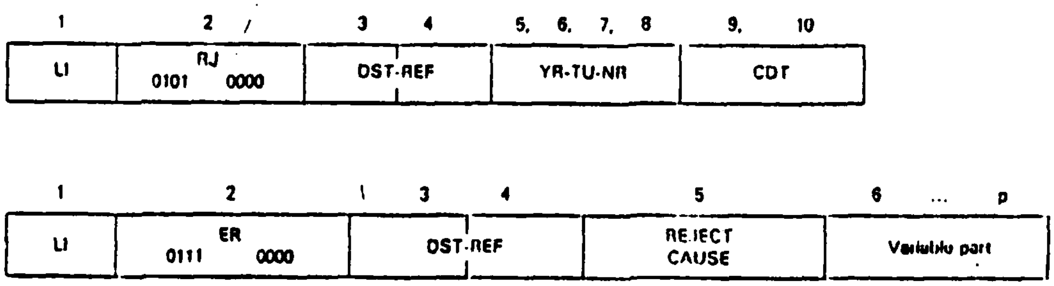


Figure 13.9 The Structure of Reject (RJ) and Error (ER) TPDU

13.2.3.7 Reject (RJ) and Error (ER)

The structures of **Reject** (RJ) and **Error** (ER) TPDUs are shown in Figure 13.9. The structure for Reject is straight forward. The "Reject Cause" in the Error TPDU gives an explanation of the reason for the error condition.

13.2.4 TC State Transition

Since the transport protocol defined in this standard is connection oriented, the state transition specified here also includes the part that deals with the network connections. The complete state transition is so complicated that it has to be explained in a state table, not in a graphic format. On the other hand, in our case, the connectionless-mode network services are used instead of connection-mode, and the state transition for the transport protocol is simpler. This will be described in next section.

14. THE SESSION LAYER

The reason for the Session Layer to be briefly described in this section is the incompatibility between the DDN TCP and ISO TP protocols. In the OSI Reference Model, the Session Layer provides necessary services for their users to organize their dialogue and to manage their data exchange. It takes full care of the session-connection establishment and release in a synchronized manner. So the Transport Layer does not repeat the function for the synchronized release of transport connection. On the other hand, in the DDN Internet architecture, there is no explicit session layer, TCP is taking the full charge for the transport connection establishment and release in the synchronized manner. If the TCP user at one side issues the CLOSE command to disconnect the TCP connection, the TCP user at the remote side is notified, the data exchange may be continued until the remote TCP user issues a CLOSE command. Both side of TCP connection are well-synchronized in this method. To match the services and functions between DDN TCP and ISO TP, the normal connection release function from the Session Layer are examined here.

14.1 SESSION LAYER SERVICES

The Session Layer provides the services necessary for co-operating presentation-entities to organize and synchronize their dialogue and to manage their data exchange.

1) establishment and release of session-connection mapped onto the transport-connection;

2) session-connection synchronization;

3) normal data exchange;

4) expedited data exchange which is free from token and flow control constraints;

5) quarantine service by which an integral number of session-service-data-units sent on a session-connection are not available to the receiving presentation-entity until explicitly released by the sending presentation-entity;

6) interaction management, to allow

   a) two-way-simultaneous (TWS);
   b) two-way-alternate (TWA); and
   c) one-way interaction.

7) exception reporting.

To support the services, following functions are performed within the Session Layer:

1) session-connection to transport-connection mapping;

2) session-connection flow control;

3) expedited data transfer;

4) session-connection recovery;

5) session-connection release; and

6) Session Layer management.

14.2 SESSION SERVICE PRIMITIVES

Table 14.1 gives a list of Session Service primitives.

The Session Service primitives for the Session connection release will be further discussed in the transport gateway design.

Table 14.1 Session Service Primitives

Service	Primitive
Session Connection Establishment Phase:	
Session-connection	S-CONNECT request/indication/response/confirm
Data Transfer Phase:	
Normal data xfer	S-DATA request/indication
Expedited data xfer	S-EXPEDITED-DATA request/indication
Typed data xfer	S-TYPED-DATA request/indication
Capability data	S-CAPABILITY-DATA req/indi/response/confirm
Give tokens	S-TOKEN-GIVE request/indication
Please token	S-TOKEN-PLEASE request/indication
Give control	S-CONTROL-GIVE request/indication
Minor sync-point	S-SYNC-MINOR req/indi/response/Confirm
Major sync-point	S-SYNC-MAJOR req/indi/response/confirm
Resynchronize	S-RESYNCHRONIZE req/indi/response/confirm
P-exception report	S-P-EXCEPTION-REPORT indication
U-exception report	S-U-EXCEPTION-REPORT request/indication
Activity start	S-ACTIVITY-START request/indication
Activity resume	S-ACTIVITY-RESUME request/indication
Activity interrupt	S-ACTIVITY-INTERRUPT req/indi/response/confirm
Activity discard	S-ACTIVITY-DISCARD req/indi/response/confirm
Activity end	S-ACTIVITY-END req/indi/response/confirm
Session Connection Release Phase:	
Orderly release	S-RELEASE req/indi/response/confirm
U-abort	S-U-ABORT request/indication
P-abort	S-P-ABORT indication

14.3 ISO STANDARDS FOR THE SESSION LAYER

The ISO 8326 standard is for the basic connection oriented session service definition.

The session services defined in this standard are performed by ISO 8327 standard: the Basic Connection Oriented Session Protocol. The session protocol is specified with respect to:

1) use of the transport service;

2) procedure elements related to Session Protocol Data Units (SPDU);

3) structure and encoding of SPDUs.

PART IV. PROTOCOL CONVERSION FROM TCP TO ISO-TP

Based on the summary over the DDN TCP/IP protocols and the ISO/OSI protocols given in previous two chapters, this section will give further discussion over the matter of matches and mismatches between the two protocol suites. The functionalities are studied for the protocol conversion tasks between TCP and ISO-TP. The methodology approaches with the protocol conversion are discussed. The rule-based general protocol conversion at the transport/network layer is concluded from the discussion.

For the gateway system design, the protocol conversion tasks are divided into subtasks, implemented by modules. The gateway specification is carefully developed with regards to easiness of understanding, and avoiding over-specification with unnecessary details. The specification is developed using a formal specification language, called ESTELLE.

15. INCOMPATIBILITY CONCERNS

The discussions in this chapter are concentrated on the different aspects of the incompatibilities, the comparison of the service provided, and the study of each incompatible items between TCP and ISO-TP. The incompatibilities between the IP and the ISO-IP are not studied here. The handling of the incompatibilities between IP and ISO-IP can be described as one to one static PDU translations, which will be discussed in the design.

15.1 DIFFERENT ASPECTS OF THE INCOMPATIBILITIES

From the view point of the functionalities, comparing the functions of TCP described in the beginning of the section 10 and the functions of ISO-TP described in Section 13.2.1, both of them provide very similar services. This condition is important because it implies that hard mismatches which could render gateway system inoperable to not exist.

From the view point of the Protocol Data Unit (PDU) structure, they are similar with suitable differences. Translation of PDUs is needed in the gateway. But this translation can be viewed mainly as a protocol-dependent part of the gateway task. It implies that those fields in the PDU related only to one particular protocol should be handled by the protocol-dependent submodule. One example is that the checksum field for different protocols can be processed separately. On the other hand, those fields in the PDU which have the common meaning for both gateway protocols but with different format should be translated cooperatively by the protocol-dependent and protocol-independent submodules. Examples of the operation codes and the control flags should be interpreted in this way.

From the view point of the interfaces between protocol layer and the service user, it has no significance here because the service user is not considered here. One example to explain this point is that the transport gateway considers only the interoperability between transport protocols, not the interface to the transport service user. The same kind of upper layer protocol is assumed in the case. The interface design in this gateway will be aimed on the interfaces between the submodules.

From the view point of the internal state transition and the peer entity interaction, conversion is a more complicated matter. One example is that ISO-TP allows the Connection Request TPDU to carry user's data, while DDN TCP does not support this option. There is no one-to-one translation to resolve the difference. There is no method to attack this problem. It can be simply handled by restricting ISO-TP user to use the option to emphasize the usage of common-subset functions, as proposed in [GROEN 86]. The problem can also be solved in a more complicated way to have the gateway to take more responsibilities. More studies are needed in attacking this kind of problem.

15.2 SERVICE COMPARISON

To be prepared for the methodology discussion for the protocol conversion, the comparison between the TCP and the ISO-TP is summarized in the Table 15.1. Further technical detail about those difference will be studied in following sections.

15.3 CONVERSION TASKS IN CONSIDERATION

In this section, the conversion tasks and the methods used for the protocol conversion will be discussed item by item. Each item represents an entry of incompatibility between TCP and ISO-TP as listed in Table 15.1.

Table 15.1 Service Comparison between TCP and ISO-TP

Function/Event	ISO-TP	TCP
TC Establish Phase:		
Connect collision	Two TC established (one way)	One TC established for given pair of socket
Addressing	Any possible structure	Static length
Expedited data	Yes, negotiable by user	No, has URGENT signal
QoS	Loosely defined, multifunctional	well-defined subset of ISO
User data	Limited length	No
Data Transfer Phase:		
User data	Octet stream	Octet stream, may be divided into TSDUs using PUSH
Urgent Signal	No	Yes
Expedited data	Precedence over normal data	No
Flow control	Explicit or implicit	Explicit
TC Release Phase:		
Orderly release	No	Yes
Abrupt release	Yes	Yes

15.3.1 TC Establish Phase

Connection Collision. In TCP, if the TC initiator, after sending the Connection Request (CR) to a destination host, receives a CR from the destination host, only one connection is going to be established, and both hosts share the connection in the communication. On the other hand, ISO-TP will try to establish two one-way connections.

In matching the difference, some researchers suggests that the ISO-TP addresses should be grouped, such as into even and odd addresses, for the outgoing and incoming calls. As far as we are concerned, it is not good idea to impose this kind of restriction on the addressing scheme. First, by examining the TCP/IP addressing scheme and the predefined TCP port addresses, it is obvious that most of the ports are defined as passive servers for various applications. It means that it is not likely to have a high probability for the connection collision. This should be true for the ISO-TP case. Secondly, if the ISO-TP host is the first CR initiator while the TCP host initiates the colliding CR, it can be absorbed by the gateway submodule at the TCP side without notifying the ISO-TP side. If the ISO-TP host initiates the colliding CR, the ISO-TP host must explicitly provide the calling address within ISO-TP addressing allowance. With this method, the connection collision can be solved by special functions in the gateway submodules.

Addressing. TCP allows only static length, fixed format port addresses, while ISO-TP allows any possible structures to be used in the addressing scheme. To resolve the incompatibility, restrictions have to be imposed on the ISO-TP users that only TCP-compatible addressing formats are allowed while calling TCP users. This requirement should be reasonable because whether the ISO-TP user is calling TCP host or calling through TCP node, the TCP address has to be used, either by calling (TP) user or translated by the gateway.

Expedited Data. ISO-TP provides services for transferring expedited data, negotiated at the beginning of connection establishment between users by selecting different Transport Service classes. TCP does not support the expedited data transferring, but it has URGENT signal used for transferring URGENT data. This difference will be discussed as one of the points in the data transfer phase.

Quality of Service. Quality of Service (QoS) is loosely defined by multifunctional control parameters in ISO-TP. In TCP, it is a well-defined subset of ISO QoS. For the same reason, restrictions will be imposed on the ISO-TP users to use only those QoS compatible with the TCP QoS.

User Data. In the ISO-TP, user data can be contained in the Connection Request TPDU with limited length. But it is not allowed in the TCP in Connection Request. This incompatibility will cause problems if the TPDU-to-TPDU translation technique is used in the gateways. In the other case, if TPDU-to-TPDU translations not used, a gateway submodule can separate the user data from the CR TPDU, store the data temporally, and compose special data TPDUs to transfer them after the connection is established.

15.3.2 Data Transfer Phase

User Data. Basically, both TCP and ISO-TP transfer user data as octet streams. In TCP, user data may be divided into segments by using PUSH signal. It is very similar to the concatenation function of the ISO Transport Layer.

Expedited data. ISO-TP provides the user the service to transfer expedited data, with precedence over normal data. TCP does not provide service, but it allows a user to use the URGENT signal to send urgent data. Basically, they are different because URGENT data in TCP allows the transfer of control information along with the user data, with or without higher priority, depending on the interpretation of the higher layers. The expedited data in ISO-TP is a measure to assure the correctness of the data transfer. It is possible to use the URGENT data of TCP network for the Expedited Data service from the ISO-TP network in the following manner. When the gateway submodule receives the Expedited Data from the ISO-TP host, it will add a predefined special segment of URGENT data to the TCP TPDU to be transferred to the destination TCP host. A Special interface at the destination host will pick up the message in the URGENT data part of the TPDU, and convey it to be the Expedited Data before sending it to the higher layer ISO software. It can be considered in the same way if the DDN application package runs in the ISO network.

Flow control. In TCP, the flow control is explicit. While in the ISO-TP, the flow control could be explicit or implicit. There are two ways to solve the problem: restrict the ISO-TP users to use explicit flow control only; or absorb the implicit flow control at the gateway submodule.

15.3.3 TC Release Phase

Orderly release. TCP provides the service for the orderly release of the connection. If the user at one end sends a CLOSE to close the connection, the TCP at the other end will notify the user, continue with data transfer if necessary, wait for the user to issue the CLOSE, and then close the connection. In ISO standard, the orderly release is considered a task for the session layer protocol. So the ISO-TP does not support the orderly release function. One of the approaches to solve the problem is to add a sublayer above the ISO-TP to include the orderly release the connection. Another approach is to resolve the incompatibility in the gateway submodule. If the TCP host issues the CLOSE request, the gateway will produce a Disconnect Request to the ISO-TP host, and continue the orderly release with the TCP host as if the orderly release has been done with the ISO-TP host. The ISO upper layers using the TCP service should be clear about this difference, and the Orderly Release is still required at the Session Layer in order to call the ISO host. The Orderly Release at the Transport Layer will be performed after the Session Layer has performed the Orderly Release and called the Transport Layer for the CLOSE.

16. CONVERSION METHODOLOGY

This section describes approaches for protocol conversion between TCP and ISO-TP. Each method has its merits and disadvantages, related to cost and performance.

16.1 CONVERSION TASKS IN GENERAL

For the purpose of protocol definition and specification, protocols are usually defined in terms of functional descriptions, procedural descriptions (operational model), structural descriptions (PDU structure), and interface descriptions.

For the purpose of protocol conversion analysis, the protocol machine can be divided into three functioning modules: interface, peer, and control protocols. The interface protocol takes care of interaction between the layer protocol with service users and underlying layer protocol support. The peer protocol is in charge of interaction between the local and remote peer entities at the same layer. The control protocol manages the internal state transition of the protocol machine. Accordingly, the protocol conversion tasks are studied with respect to these three aspects. The interface to the service user is of no importance here because the assumption that the upper layer protocols above this converted layer are the same. There should be no change made in the interface to the underlying supporting protocol.

The peer protocol conversion includes the protocol data unit translation. For those data fields which is only related to one side protocol in the conversion, they should be processed locally to that protocol. Other data fields also need to be translated in certain strategies. The control protocol conversion is of much more complicated matter which will be discussed in more details later.

16.2 DIRECT CONVERSIONS

In the first serious attack of the protocol conversion problem at the transport layer between DDN TCP and ISO-TP was reported by Groenbraek in [GROEN 86]. In this treatise, the direct conversion method is used. Based on the study of the protocol differences, a state-to-state, PDU-to-PDU conversion method is used. The advantage of this approach is that the conversion procedure looks straight forward. It contains the direct conversion of all states from one protocol to another. However, this approach imposes disadvantages on the implementation.

1) Unnecessary repetition of similar details;

2) No consideration to the extension of this work to other protocols;

3) Imposes heavy restrictions upon users which results in a critical drawback to performance. This point will be further discussed below.

16.3 RESTRICTION-UPON-USER APPROACH

This approach has been discussed in several papers in [GREEN 86], [GROEN 86], etc. It was recognized in these papers that the first step in the protocol conversion is to find common set of the services, and then convert the functions related to these services. The principle behind this approach is obvious. The conversion is performed between the services of the users.

This approach imposes two serious disadvantages. One of them is the restriction upon users. By the conversion only over the common set of the services, it restricts the users from using those services which are not in the common set of services. Since it is impractical to modify the communication software at upper layers on all the hosts, it puts the burden on those users using the service through gateways. Another disadvantage is that this approach does not consider those mismatches which need more intelligence to be solved. Some examples will be given in following section, and further discussion will be given in next chapter.

16.4 RULE-BASED METHOD

In this approach, those services with minor incompatibility problems will be solved with a little more intelligence in the gateway system. For example, TCP does not allow the Connection Request (CR) TPDU to carry any user data. In case that user data are contained in the Connection Request TPDU from ISO-TP, instead of not allowing user to do that, the gateway can divide CR into Connection Request TPDU and the Data TPDU for the TCP network. The reasonability for this approach is on the TCP side and the ISO-TP side must keep their own Sequence Number. In this way, extra data TPDU will not disturb the normal operation. The acknowledgement for the extra data TPDU will be consumed at the gateway because it does not need to be sent to the ISO-TP network. The gateway needs to keep record of the extra TPDUs it creates.

Another example is the PUSH function in TCP. The PUSH function in TCP means that when a user indicates the PUSH flag in the SEND data call, the TCP module should not wait for further user data to be concatenated for efficiency, but rather send the user data right away. This function requests only immediate delivery of the user data. It can be handled by either ignoring the PUSH flag and sacrificing a little performance, or by using other control functions such as Expedited Data in the ISO-TP network. The rule-based method can be approached so that the knowledge about the characteristics of the service and the knowledge about the way to make up the difference and to solve the incompatibility are used in handling different service functions.

16.5 MODULAR SYSTEM DESIGN

With the consideration discussed in the last section, the gateway system design should take the modular system design approach. The major point is to divide the gateway system into three parts: the protocol-dependent modules, the general purpose linker module, and the interface modules to connect them together. The superset of the service provided by different transport protocols, instead of subset of common service, should be considered. The results then are used for designing the data and control structures in the general purpose linker module. The rule-based tasks will be handled in the interface module to resolve the difference between the general purpose linker module and the protocol-dependent modules. The details of the design will be explained in the following sections. A similar approach was followed in the testing of functional specifications of a generic gateway [MART 88].

17. FUNCTIONAL MODULES

In this chapter, the modular structure of the gateway system to interconnect the TCP/IP and the ISO/TP is discussed. The functions of each module are described based on the system design.

17.1 THE MODELS OF THE PROTOCOL CONVERSION

This section contains the models for protocol conversion. The two models used include the Message Translation Model and the State Translation Model.

17.1.1 The Message Translation Model

The Message Translation Model is shown in Figure 17.1. This model deals with the message translation, which corresponds to the handling of the PDU subprotocol and the interface subprotocol. The handling of the PDU subprotocol is obvious in this case. All the PDUs from the local network, say P′, are temporary stored in the packet buffer, and processed in the local-network interface module to check the validity. The local-protocol dependent parts of P′ are stripped off and the original PDU P′ are translated into P, depending on the design. Examples are the correctness of the checksum, the validity of the address, the sequence numbers and window parameters, and the legitimacy of the packet control commands with regard to the FSM state in the protocol module. A record of the connection control information of local-protocol dependent parts will be kept in the local interface module. The data and the control information with common interests will be passed over to the linker module, where the control information is kept for reference for follow-on communications. The data and the necessary control information, in the format of P, will be then relayed to the remote interface module. In the remote interface module, similar work in the local module will be reversed to make up the PDU" from P, by adding in the protocol dependent control information. The PDU P" will then be sent through the service provided by the underlying layer to the remote network. Whether the packet will be kept in the buffer or not until acknowledgment is received is completely

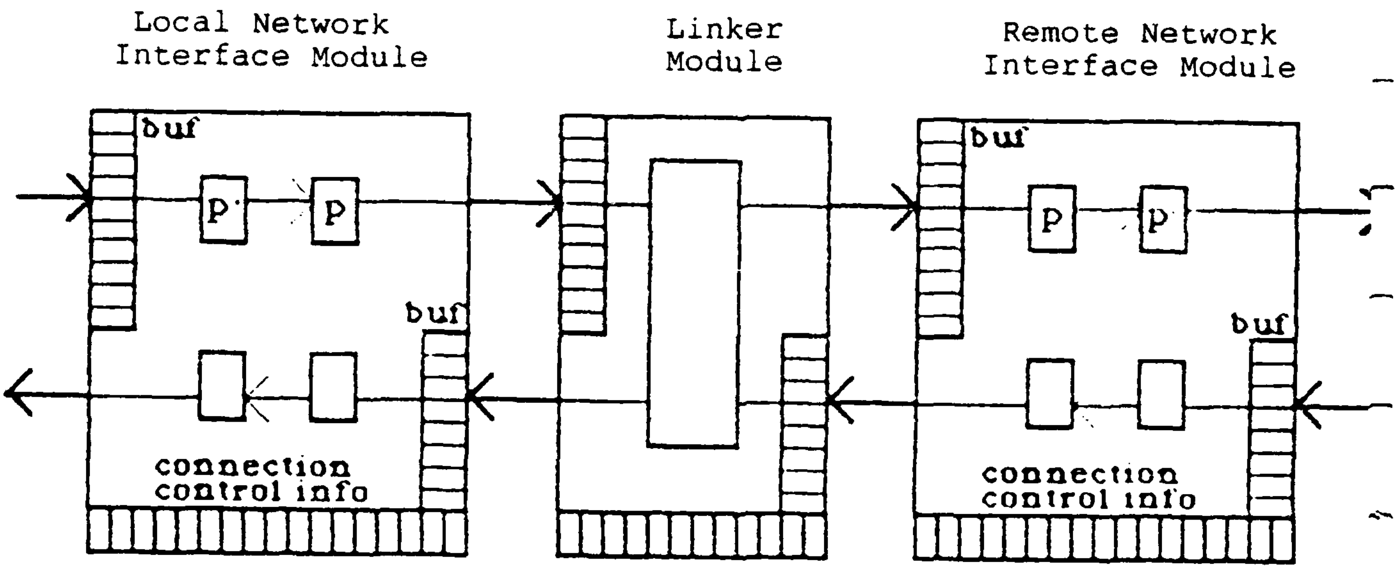


Figure 17.1 **The Model of Message Translation in Gateway**

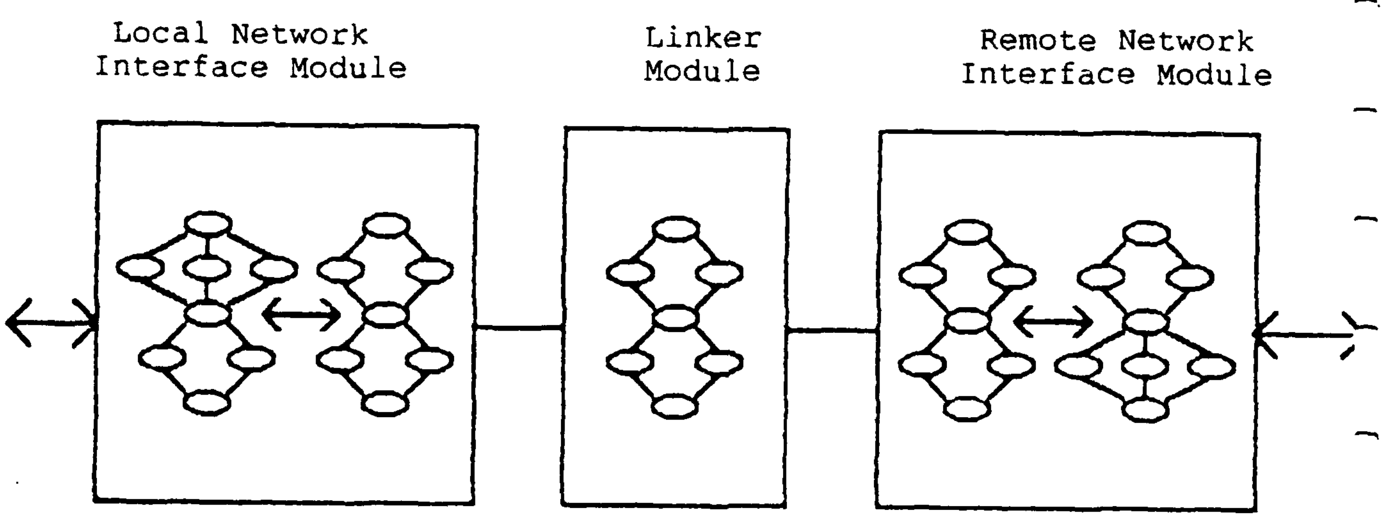


Figure 17.2 **The Model of State Conversion in Gateway**

dependent on the gateway strategy.

The handling of the interface subprotocol is done in the process of the communication. The interface subprotocol keeps the interface compatibility between this layer and the underlying layer. The interfaces between the local module and the linker and remote modules are implementation dependent. Though, the integrity of the service interface between this layer and the upper layer should still kept in consideration.

17.1.2 The State Conversion Model

Another model of the gateway task is shown in Figure 17.2. This model deals with the state conversion, which corresponds to the handling of the internal state transitions and the peer entity interaction.

The local and remote protocol-dependent modules in this model plays two major functions. One function is to maintain the internal state transitions as defined by its own protocol. The other function is to convert the state defined in the local protocol into the common state handled by the linker module, and vice versa. The set of common states is the super set of all states in the protocols. The linker module in this model is merely an interface, it defines the super set of the common state. It records the state of each communication channel during its operation using this super set of state. Since each set of states of individual protocol is the subset of the common state set, the difference between individual subset and the common set is made up of protocol-dependent modules. If there exists a one-to-one relationship between the pair of local protocol states and the common states, only the name translation is needed. If there is no such one-to-one relationship between the pair of states, the protocol-dependent module will be in charge of the translation between the subset states and the common set states. All these state processing should be confirmed with the peer entity interaction specification of the local protocol with respect to the local network.

17.2 SYSTEM STRUCTURE

In this section, the gateway layer and the system designs are discussed based on the basic models described above.

17.2.1 The Layer Design

In the gateway architecture designed here, the structure of each layer in the gateway is composed of both models discussed in last section. The common linker module and the protocol-dependent modules are in charge of both the message translation and the state conversion at the same time.

As shown in Figure 17.3, the linker module defines and uses the set of common states and the common data structure for the communication session control and for the data packets transfer to communicate with the protocol-dependent modules. The state of the communication session and other control information are kept in record in the linker module. The data packets are relayed between the protocol-dependent modules. The protocol-dependent modules can be further divided into two submodules. One of them is the local protocol-dependent submodule which takes care of the packet validity and state legitimacy checking and the communication session status maintenance according to local protocols. The other is the interface submodule which takes care of the translation of the packet structure and the state transition between local protocol and common definition of the linker module.

The local protocol-dependent submodule takes the full responsibility of the interface subprotocol with the underlying layer, the PDU correctness checking, and the peer entity interaction with the communicating nodes in the local network. For example, it automatically establishes the lower layer connection when the connection is requested at this layer. The lower layer provides connection-oriented service. It also encapsulates and decapsulates the packets with the protocol control information of the layer, such as sequence number, window control, etc. It will send back reasonable responses and change its own state when service is requested by the other nodes in the network.

The interface submodule translates the packet data structures into the common data structure, along with the translation of the control command, options. The protocol-dependent parts of packets will be discarded in the translation. It is necessary to reduce the amount of the information to be converted to reduce the delay. On the other hand, when the data packets and the control information passed over from the linker module which are not supported by the local protocol, it is also the interface submodules responsibility to resolve the incompatibility.

17.2.2 The System Design

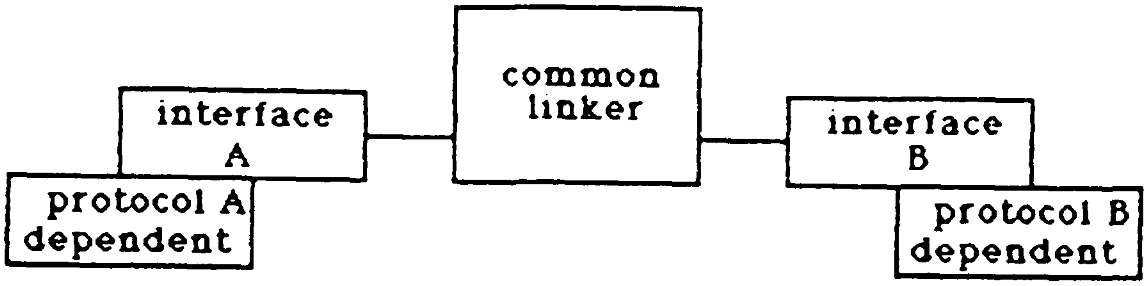


Figure 17.3 Moduled Design of the Gateway Layer

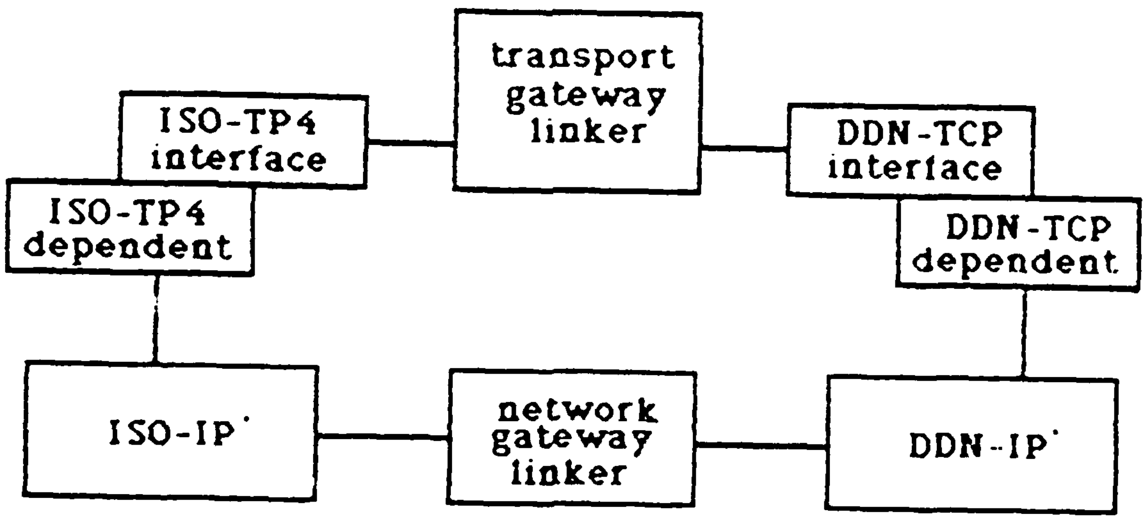


Figure 17.4 Moduled Design of the Gateway System

The Transport Gateway designed in this project is concerned with both the Network Layer and the Transport Layer, as shown in Figure 17.4. As a case study, the TCP and the ISO-TP4 with the connection-oriented reliable virtual circuit service will be considered in the transport layer, and the IP and the ISO-IP with connectionless datagram service will be considered in the Network Layer. It is possible to use only one linker module for both the Transport and Network Layers, but it is against the concept of layered architecture and modular design. So, separate linker modules are used for each layer in the case. Each linker module takes care the communication in its own layer.

Not much control information are recorded in the Network Layer linker module, because both DoD IP and ISO-IP provide the datagram service in which datagrams are not logically related to each other. On the contrary, there is more common control information in the Transport Layer linker module, such as addresses, state of the connection, etc. The data structures and algorithms are designed to reduce the amount of information to be transferred between modules.

Better modularity is achieved by allocating the queue of the incoming and outgoing packets inside the interface submodule, and having the linker module in charge of transferring packets between the interface submodules of individual local protocols. There will be one queue associated with each direction of data flow in the interface submodule. The interface submodule reads packets in the incoming buffer queue from the linker module, and deposits packets into the outgoing buffer queue to the linker module, and modifies the pointers accordingly. The linker module treats the buffer queues of all interface submodules equally. It reads packets from the outgoing buffer of one interface submodule and deposits packets into the incoming buffer of another interface submodule. Using this strategy, the gateway is capable of converting more than three sets of protocols without critical modification of the system design. More technical issues in the design will be discussed in following sections.

17.3 PROTOCOL CONVERSION IN THE NETWORK LAYER

The Network Layer of the gateway in design translates the DoD Internet Protocol (IP), referring to Chapter 7, to and from the ISO-IP protocol, referring to DIS 8473 in Section 12.2. Both protocols provide the connectionless datagram service.

17.3.1 Protocol Conversion Tasks

In the DoD IP and the ISO-IP, the following data fields and related functions are protocol-dependent. They are handled inside the protocol-dependent submodules, not translated during the communication:

DoD IP	ISO-IP
IP Version	IP Version
Protocol	
Header Checksum	Header Checksum
	network Layer Protocol ID

All the other data fields listed below need to be translated and transferred to the other side in the communication:

DoD IP	ISO-IP
Internet Header Length	Length Indicator
Type of Service	Type Code
Total Length	PDU Segment Length
Identification	Data Unit Id
Flags	Flags
Fragment Offset	Segment Offset
Time to Live	PDU lifetime
	Source Address Length
Source Address	Source Address
	Destination Address Length
Destination Address	Destination Address
Options	Options

Among the parameters, the Internet Header Length in DoD-IP and the Length Indicator in ISO-IP are the length of the headers in 32-bit word and byte respectively. The original parameters have no significance to the other protocol sets because the header formats are different. It should be modified to indicate the length of the option. Translation is needed between the Type of Service, the Flags, and the Options in DoD-IP and the Type Code, the Flags, and the Options in ISO-IP.

The Total Length in DoD-IP and the PDU Segment Length in ISO-IP should be adjusted according to the justification of the header length. Minor modifications are needed for the Identification, and the Fragment Offset in DoD-IP and the Data Unit Id, and the Segment Offset in ISO-IP, such as the modification on the size of the field. The time unit for the Time to Live in DoD-IP should be changed from seconds into half-seconds used in the PDU Lifetime in ISO-IP.

For the address fields, the ISO-IP is restricted to use the address formats which can be converted into the CLASS A, B, and C address formats of the DoD-IP. Translation is needed between the address format.

17.3.2 Protocol-Dependent Submodule

The protocol-dependent submodule is the front-end interface which receives and delivers packets to and from the sub-network layer of the local network. It checks the validity of the packets, records the status into the statistics, and strips off the fields with only local network significance. It should confirm with the local protocol definitions about the interface subprotocol with the lower layer and the peer entity interface subprotocol with the communicating entities in the DoD-IP and ISO-IP.

The statistics keeping should be emphasized here due to the fact that the Internet Control Message Protocol (ICMP, referring to Chapter 8) is actually implemented along with DoD-IP. ISO-IP includes some error control functions inside. The conversion of the difference is handled by the combination of the protocol-dependent and the interface submodules.

17.3.3 Interface Submodule

The translation of the data structure of packets is one of the major functions of the interface submodule. As discussed in previous sections, all the fields with data significance but in incompatible format, needed to be translated. This submodule also manages the buffer queues to be accessed by the common linker module. After the buffer queues are allocated, it should notify the common linker module about the location and the control information of the queues. The packets translated from DoD-IP to the common data structure are deposited in the outgoing queue. The packets picked up from the incoming buffer queue are translated back to the DoD-IP.

Another major function of this submodule is the translation of the DoD-ICMP packet to the ISO-IP Error Packet. This task should be carefully coordinated with the protocol-dependent submodule.

17.3.4 Common Linker Module

The tasks performed in the common linker module are rather simple. It defines the common data structures used in the communication between the network-dependent interface submodules. After initialization, it collects the control information about the Network Layer protocols. This includes the locations and the pointers of the buffer queues. The remaining tasks are checking the buffer queues of each interface submodule when interrupts happened, relaying the packets in the buffer if the buffers are not empty.

17.4 PROTOCOL CONVERSION IN THE TRANSPORT LAYER

The Transport Layer of the gateway is designed to translate between the DoD Transmission Control Protocol (TCP) and the ISO-TP Class 4. Both protocols provide connection-oriented reliable virtual circuit service. It is clearly defined that TCP uses the services provided by DoD-IP for the underlying Network Service. The ISO Transport Protocol Specification defined in IS 8073 is based on the connection-oriented Network Services. Adjustment is needed in this case to refer to the DIS 8602, Protocol to Provide the Conncetionless-mode Transport Layer Service. The case of protocol conversion between the DoD User Datagram Protocol (UDP) and the ISO-TP Class 1 could be a much simpler case of the TCP to TP4 conversion.

17.4.1 Protocol Conversion Tasks

The protocol conversion tasks for this layer includes the Transport Protocol Data Unit translation and the FSM state processing.

The following fields and related functions are protocol-dependent between TCP and TP-4 which do not need translation.

DoD TCP	ISO TP-4
Checksum	Checksum
Sequence Number	Sequence Number (TPDU-NR)
Acknowledge Number	Acknowledge Number (YU-TU-NR)
	Subsequence Number
Window	Flow Control Confirmation

Checksum will be added for the TPDU just before sending it to the Network Layer, and it will be checked first after receiving the TPDU from the Network Layer. Different protocols have different interpretation of the sequence numbering. The sequence number in the TCP counts the bytes of user data transferred by TCP, and the sequence number in ISO-TP stands for the numbering of TPDUs. It is more reasonable to have the protocol-dependent submodules to handle the sequence and acknowledge numbers and/or window control independently.

The other parameters have their own definitions and formats for different protocol suites, and they need to be translated between the transport entities:

DoD TCP	ISO TP-4
Source Port	Calling TSAP Identifier
Destination Port	Called TSAP Identifier
Data Offset	Length Indicator (LI)
Flags	TPDU Code
Urgent Pointer	
Options	Options
	Source Reference (SRC-REF)
	Destination Reference (DST-REF)
	Disconnect Reason (REASON)
	Reject Cause (REJECT CAUSE)
	TPDU size
	Version number
	Protection
	Alternative Class
	Acknowledge time
	Throughput
	Residual Error
	Priority
	Transit Delay
	Reassignment Time

Among the parameters, the Source and Destination References in the ISO-TP should be so restricted that it can be represented by the Source and Destination Ports in TCP with the length of 16 bits. The Data Offset in TCP should reflect the length of the options. The Length Indicator in ISO-TP should reflect the length of the variable part of the TPDU. Translation is needed between the Flags and the options in TCP and the TPDU Code, and the Options in ISO-TP. The other optional control parameters provided by ISO-TP do not have the counterparts in the TCP. They should be handled under reasonable policies.

17.4.2 Protocol-Dependent Submodule

The protocol-dependent submodules interface to the Network Layer to send TPDUs to and to receive TPDUs from the local networks. It checks the validity of the TPDUs by checking the checksum and the legitimacy of the TPDU Code, records the status into the statistics, and strips off the fields with only local protocol significance. It maintains the state transition of the FSM of the transport entities. It should meet the local protocol definitions about the interface subprotocol with the Network Layer, and the peer entity interface subprotocol with the communicating entities in the other TCP and ISO-TP nodes.

17.4.3 Interface Submodule

The translation of the data structure of TPDUs and the Transport FSM state are the major functions of this submodule. As discussed in previous sections, all the fields with data significance but in incompatible format needed to be translated. The submodule also allocates and manages the TPDU buffer queues to be accessed by the common linker module.

17.4.4 Common Linker Module

The procedures executed in this module are rather direct. It collects the control information about the locations and the pointers of the TPDU buffer queues after they are initialized. The control information is used for accessing and controlling the TPDU relay. The control information about each communication session should also be maintained in the common linker module so that the supervision of the virtual circuits can be implemented.

17.5 *DESIGN TOOLS*

The most important thing in the communication system design is to handle the concurrence in an unambiguous way. In order to select the right tool for specifying the gateway system, the current protocol description techniques will be reviewed first.

In recent years, a lot of efforts have been put in the research of the Formal Description Techniques (FDTs). Generally speaking, they are the methods to define the system behavior without using a natural language so that the system can be analyzed and interpreted unambiguously. They are important tools for the design, analysis and specification of information processing systems so that the system descriptions can be produced in a self-contained, complete, consistent, precise, concise and unambiguous way. Two FDTs defined by ISO are ESTELLE and LOTOS.

ESTELLE is a second generation FDT based upon the extended Finite State Machine (FSM) model which is considered to be closer to human understanding of protocol. A specification in ESTELLE is comprised of a set of modules which communicate with each other. Modules are specified as extended FSMs. ESTELLE is a procedural technique. Facilities in ESTELLE consist of a set of extensions to PASCAL, specifying transitions in PASCAL, introducing algorithm details, thereby often over-specifying and diminishing the applicability of FDT. The communication in ESTELLE is done by asynchronous message passing FIFO queue buffering.

LOTOS (Language Of Temporal Ordering Specification), on the other hand, describes the system by defining the temporal relation between events in the externally observable behavior. One of the two components deals with the description of process behaviors and interactions, based on the non-procedural predicate calculus and modified Calculus of Communicating System (CCS). The other component deals with the description of data structures and value expressions, based on the Abstract Data Type (ADT) language ACT ONE.

LOTOS can express the concurrence and the functional abstraction unambiguously. But unlike the protocol specification with the emphasis on the description of the externally observable behaviors, the gateway design are more emphasized with clear description and easy understanding of the design. Since the procedural description with state transition in FSM, like the one in ESTELLE, are closer to the human understanding, it is preferred with moduled design to the non-procedural predicate logic in LOTOS. On the other hand, the functional abstraction using rule based design will be considered as necessary to avoid over-specification. For the data abstraction, the modified ADT with less algebraic specifications but more state transition descriptions will be used so that the mechanism can be easily understood and constructed.

18. SYSTEM ARCHITECTURE AND COMMON DATA STRUCTURE

To get a better view and understanding of system and better understanding of the gateway design, we start the design from the system structure, the common data structures, and the TP-4 dependent submodule. We approach the problem in this way in order to give detailed understanding of how the transport services are provided through the internal mechanism. Then, the transport linker module, the interface submodules for both TP-4 and TCP, and the TCP dependent submodule will be designed. The modules in the network layer will be designed in the last section.

In this section, the transport gateway system structure will be described first. The data structures and module descriptions will then be designed in correct ESTELLE order. The transport gateway specification in ESTELLE is included in Appendix A.

18.1 SYSTEM ARCHITECTURE OF TRANSPORT GATEWAY SPECIFICATION

The Transport Gateway system is composed of two layers with several modules in each layer, as indicated in Figure 18.1 below.

The ISO-IP and DoD-IP dependent submodules directly communicate with local networks. All the other modules communicate with each other by exchanging messages.

One difficult problem in the approach is how to convey the network control information (Network PDU header) between the network module which receives incoming PDUs and the network module which sends outgoing PDUs. There is no such kind problem in an ordinary node because the upper layer (Transport Layer) user specifies the network control parameters for sending data when requesting the network services. The network control information in the received PDU header is decapsulated before transferring the data part to the Transport Layer. In the case of Transport Gateway, the network control information are lost when the PDUs decapsulated. Certain scheme must be worked out to overcome this problem. It is not good idea to attach the network control information with the transport layer data because it is against the layered concept. It is possible to have the network

```
+------------------------TRANSPORT-LAYER-------------------------+
|+---------------+      +---------------+      +---------------+|
||     TP-4      |      |   Transport   |      |      TCP      ||
||   Interface   +------+    Linker     +------+   Interface   ||
||   Submodule   |      |    Module     |      |   Submodule   ||
|+---------------+      +---------------+      +---------------+|
||     TP-4      |                             |      TCP      ||
||   Dependent  .|                             |   Dependent   ||
||   Submodule   |                             |   Submodule   ||
|+-------+-------+                             +-------+-------||
+----------------------------------------------------------------+
         |                                             |
+-------------------------NETWORK-LAYER--------------------------+
|+-------+-------+       +-------------+       +-------+-------+|
||    ISO-IP     |       |   Network   |       |    DoD-IP     ||
||   Interface   +-------+   Linker    +-------+   Interface   ||
||   Submodule   |       |   Module    |       |   Submodule   ||
|+---------------+       +-------------+       +---------------+|
||    ISO-IP     |                             |    DoD-IP     ||
||   Dependent   |                             |   Dependent   ||
||   Submodule   |                             |   Submodule   ||
|+-------+-------+                             +-------+-------+|
+----------------------------------------------------------------+
         |                                             |
```

Figure 18.1 **Transport Gateway System Architecture** modules to communicate directly with each other for the control information, but a one-to-one relationship has to be kept between the transport layer data and the network control information.

One approach to solve the problem is to have the TPDU carry a token so that the network module on the other side can use the token to retrieve the network control information. The token is composed of two elements: the Network Module Id to identify the initiating network and protocol type, and the Message Id to locate the particular PDU. The Network Module Id is used to identify the buffer pool for the retrieval operation. The procedure of the communication works as follow:

a) The network module receiving the incoming PDUs will send the network control information separately to the network module on the other side, along with the local Network Module Id, and Message Id;

b) The Network Module Id and the Message Id will also be kept in the TPDUs in two special fields;

c) When the TPDU reaches the network module at the other side, the related network control information will be picked up by matching the Network Module Id and the Message Id.

This scheme will be reflected in the common data structure definition in next section.

18.2 COMMON DATA STRUCTURES

This section of the system design specifies the global control variables and the common data structures used in the message exchanging. The data structures defined here are for those messages exchanged between different modules at the system level. For example, the ISO-IP PDUs are exchanged between the ISO-IP module and the external world which is considered at the system level, while the interpretation of the PDUs is done inside the module. The data structure of the PDU is defined at the system level, while the interpretation of the values of the data fields is done inside the module.

The way the data structures and the interfaces are defined here is quite different from the formal protocol specifications. For the formal protocol specification, the important thing is to define the interface so clearly and unambiguously that it can be easily understood. But it cannot be done in the exactly same way in the gateway specification because it could make the design and implementation more difficult. So in some places, the data structures and the interfaces are defined in a way of message format oriented and implementation oriented. When the underlying Network Layer service is connectionless, only one kind of data format is needed for the communication with Network Layer.

18.3 TRANSPORT GATEWAY SPECIFICATION

The commented specification for the transport gateway is enclosed in Appendix A.

19. NETWORK LAYER DESIGN

In the NETWORK_LAYER module, the common linker module discussed previously is included as a part of the NETWORK_LAYER module definition with common data structures defined and submodule interaction handled. The submodules describe the functionalities and the interfaces for ISO-IP and DDN-IP respectively.

19.1 SOME RELATED PROBLEMS

19.1.1 Handling of Segmentation

One point which is worth to mention here is that the segmentation information in the PDU header are not necessary transferred over the gateway. The segmentation information includes the flags, the Unit Id and Total Length, the Segment Offset, and the Segment Length. The segmentation is done on the TPDU basis for those TPDU with over large size. segments are reassembled back to original TPDU before sending it back to the receiving Transport entity. It is absolutely necessary because the protocol conversion is at the Transport Layer which needs the complete TPDU, not some segments of it. After the TPDU goes through the gateway, the TPDU will be segmented by the sending network entity if necessary, but it is under the complete control of that network entity. Original segmenting information has no significance here.

19.1.2 Handling of Routing

One of the important issues in the Network_Layer Gateway here is the handling of routing. In the case of ordinary Network_layer gateway, the routing information can be achieved by packet-to-packet translation. In our case of the Transport_Layer gateway, we have the special situation that the gateway can "make up" some packets in responding to some packets just received. More complicated mechanism are needed to solve the problem:

1) The Common Linker will translate the routing information, but both ISO_IP and DDN_IP submodules need to keep records of the routing information;

2) It is necessary to encourage the usage of both source routing and record routing, so that returning routing information can be found and converted to;

3) In order to find the routing information for those "made up" packets, the IP submodules will always keep the routing information updated by referring to current received packet. The "made up" packets from Transport Layer will have IP address only, but the routing information will be assembled by the IP submodules.

4) Certain measures are required to keep the amount of recorded information down.

Following strategies are used in our design:

1) The normal one-to-one translated messages will be assigned with a non-zero "message_id" by the receiving IP submodule before sending to the Transport Layer and sending over through Network Layer Gateway. Using "message_id" will ease the task to assemble the data portion translated by the Transport Layer Gateway with the network routing information and some other control information.

2) When the IP submodule receives a PDU with zero "message_id", it knows that it is the "made up" PDU from the Transport Layer. The IP module will find the communicating pair by matching up the source and destination IP addresses, and pick up the newest routing information for that pair to be assembled into the "made up" PDU.

19.1.3 Handling of QoS Parameters

By the standard protocol specifications such as ISO-TP (ISO 8073), the Quality of Service (QoS) parameters are passed from the Transport Layer to Network Layer for service request, and from the Network Layer to the Transport Layer for service indication. But here in the case of the Transport Layer Gateway, there is no such user of the Transport Layer to specify the QoS requirements or to accept the QoS parameters from peer entity. So, there is no need to pass those control parameters between the Transport and Network Layers, but to translate and to pass through the Common Linker at the Network Layer. This concept is reflected in the message format design and handling.

19.2 NETWORK_LAYER MODULE SPECIFICATION

The commented specification for the NETWORK_LAYER module inside the transport gateway is enclosed in Appendix A.2.

20. TRANSPORT LAYER DESIGN

In this chapter, some special problems related to the protocol conversion at the Transport Layer are discussed, followed by the specification of the Transport_Layer module of the Transport Gateway.

20.1 SOME RELATED PROBLEMS

20.1.1 Classes of Service Allowed through Gateway

Considering the fact that the DDN TCP provides the reliable end-to-end virtual circuit service which is equivalent to the CLASS-4 service of the ISO TP, it is reasonable to restrict the communication through the gateway to the same type of service, i.e. TP-4. By imposing this consideration, it means that when the gateway tries to establish the connection initiated by the TCP node. It will request the CLASS-4 service at the ISO network side. On the other hand, the gateway will accept only the connection requests from the ISO network which requests CLASS-4 services. ISO TP conformance requires that when CLASS-4 is implemented, it shall also implement the CLASS-2. The CLASS-2 is not treated here.

20.1.2 Internal FSM Modifications

As the special characteristics of the transport gateway implementation, there is no Transport Layer user and the interface. The state transition model is no longer kept the same as the transport protocol implementation. Only the TPDUs and the peer entity interactions are kept the same as defined in the original protocols respectively. Examples of the state transitions for connection establishment and connection release for the ISO TP-4, TCP, and for the transport gateway are shown in Figures 20.1 to 20.8.

Both ISO TP-4 and DDN TCP use three-way handshaking in the connection establishment, as shown in Figure 20.1 and 20.2. Three TPDUs are equivalent between two protocols: connection request, connection confirm, and the acknowledge. The difference is that the TP-4 user of the receiving node in ISO network will receive a TCONind after the Transport Protocol Machine (TPM) receives the Connection Request, and needs to issue a TCONresp to confirm; while the TCP user of receiving node in DDN needs to issue a passive OPEN in advance to ready for incoming call. In the case of the Transport Gateway, the state transitions, shown in Figure 20.3 and 20.4, are neither the same as the one in ISO TP-4, nor the one in TCP. It always uses three state transition steps, and the transient states are different between the TP-4 initiated call and the TCP initiated call. It is critical to maintain the same message exchanging and peer entity interaction between the two end nodes in ISO network and DDN.

In the case of Connection Release, as shown in Figure 20.5 and 20.6, the TPM of the TCP in the DDN plays orderly release. This means that the connection is released upon the agreement between two TPM users at both side, while the TPM in the ISO network releases the connection upon the user request from one-side only. As discussed earlier, there could be several strategies to resolve the difference. If the DDN applications are supposed to run in both ISO and DDN, the TP-4 in the ISO network should include the orderly release function in the sublayer. If the ISO applications are supposed to run in two networks, the orderly release in the TCP can be simplified because the ISO Session Layer includes the orderly release function. The second case is considered in the design here. As shown in Figure 20.7 and 20.8, the simplification is done inside the Transport gateway to resolve the difference, while the message exchanging and the peer entity interaction between two end nodes are still kept confirmed with original protocols.

The state transition for the Connection Establish and the Connection Release inside the Transport Gateway are shown in Figure 20.9 and 20.10.

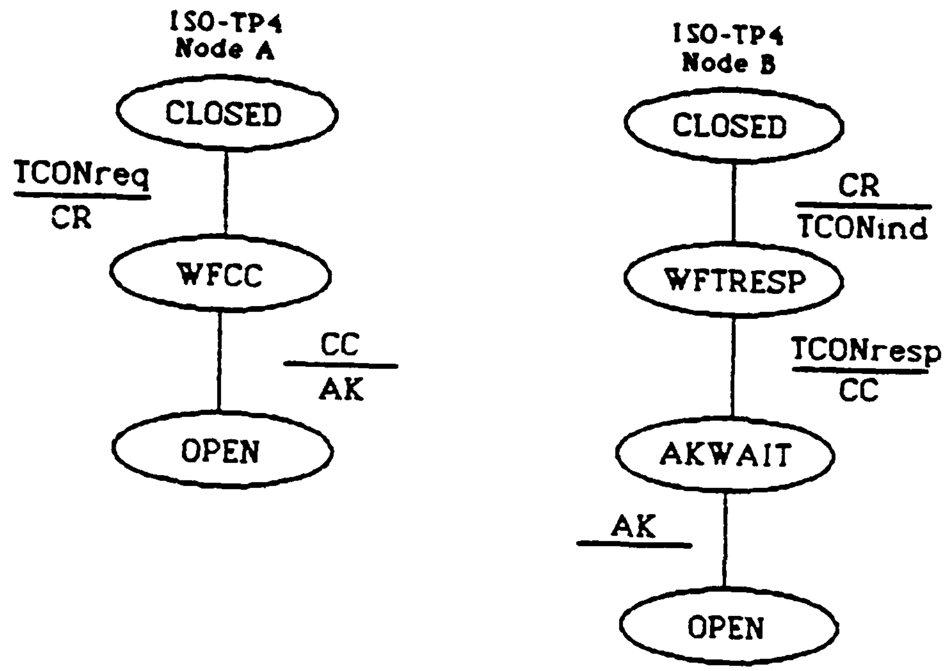


Figure 20.1 State Transition: ISO-TP4 Connection Establishment

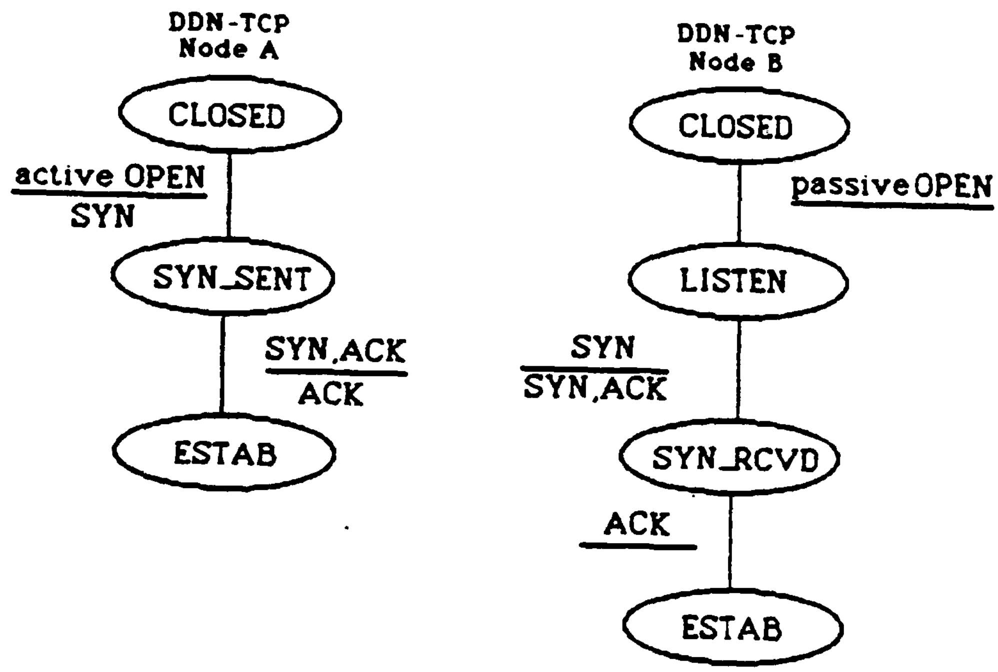


Figure 20.2 State Transition: DDN-TCP Connection Establishment

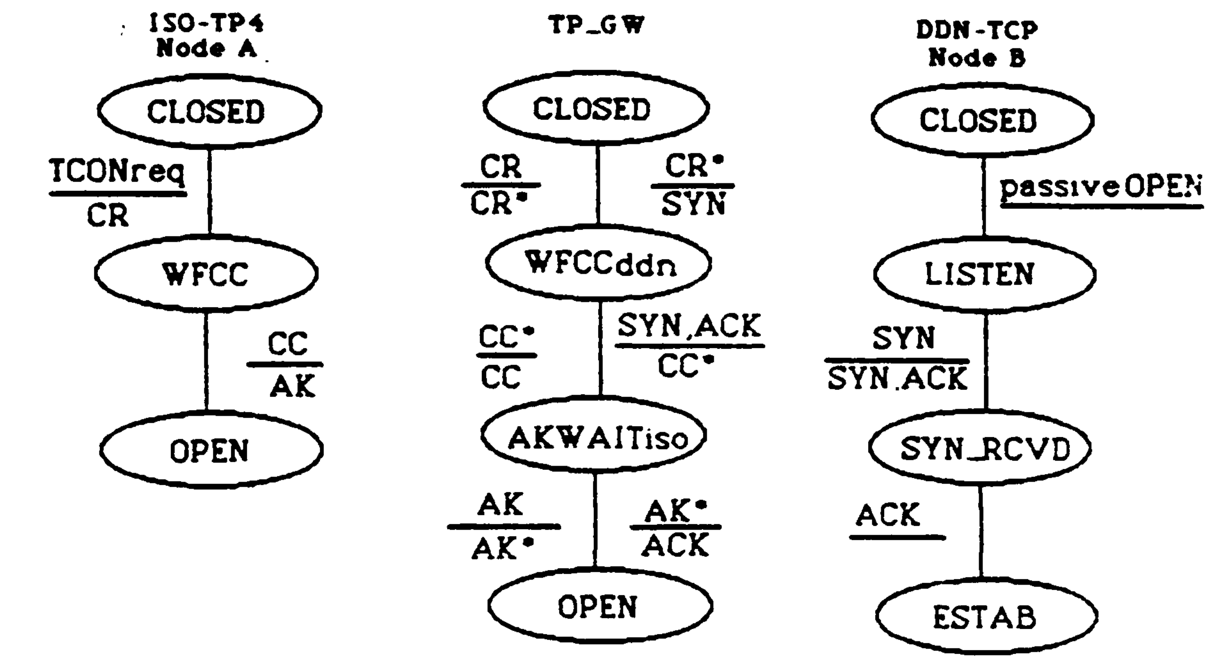


Figure 20.3 TP-Gateway Connection Establishment Initiated by ISO-TP4

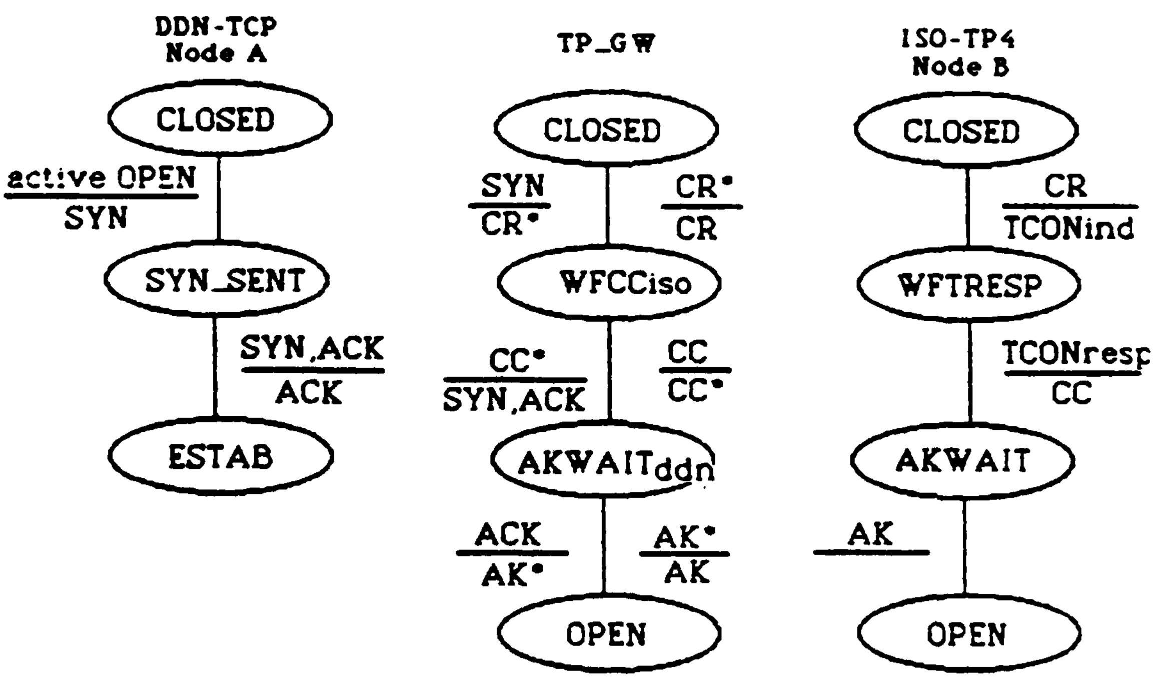


Figure 20.4 TP-Gateway Connection Establishment Initiated by DDN-TCP

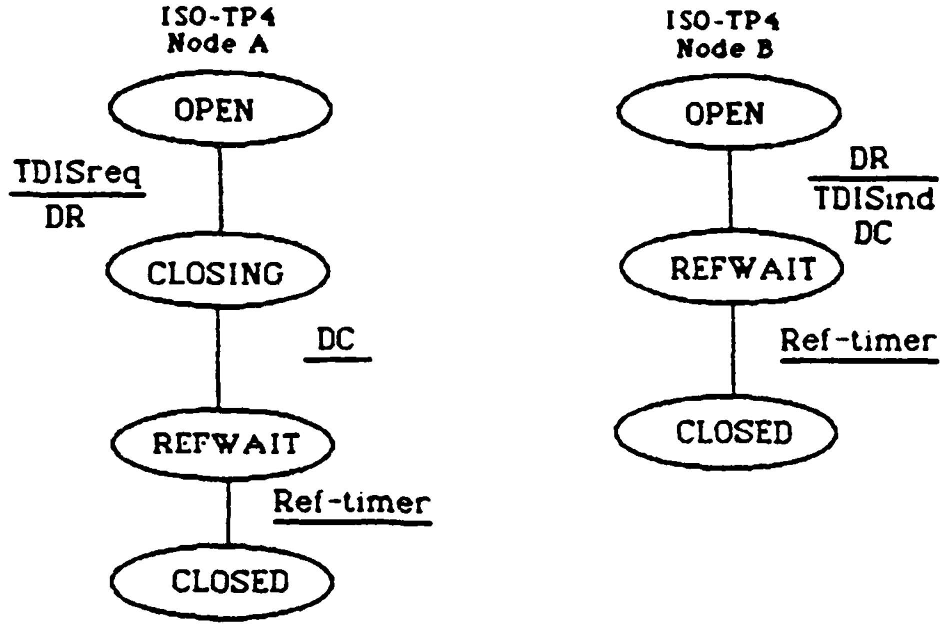


Figure 20.5 State Transition: ISO-TP4 Connection Release

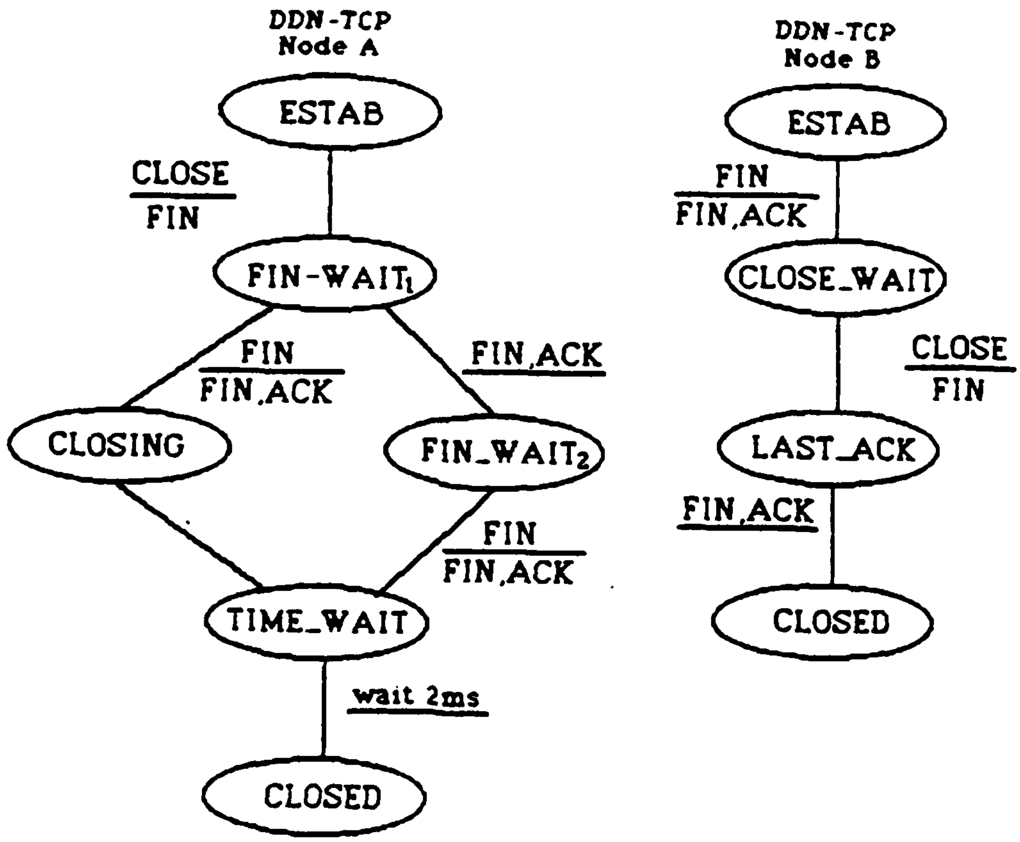


Figure 20.6 State Transition: DDN-TCP Connection Release

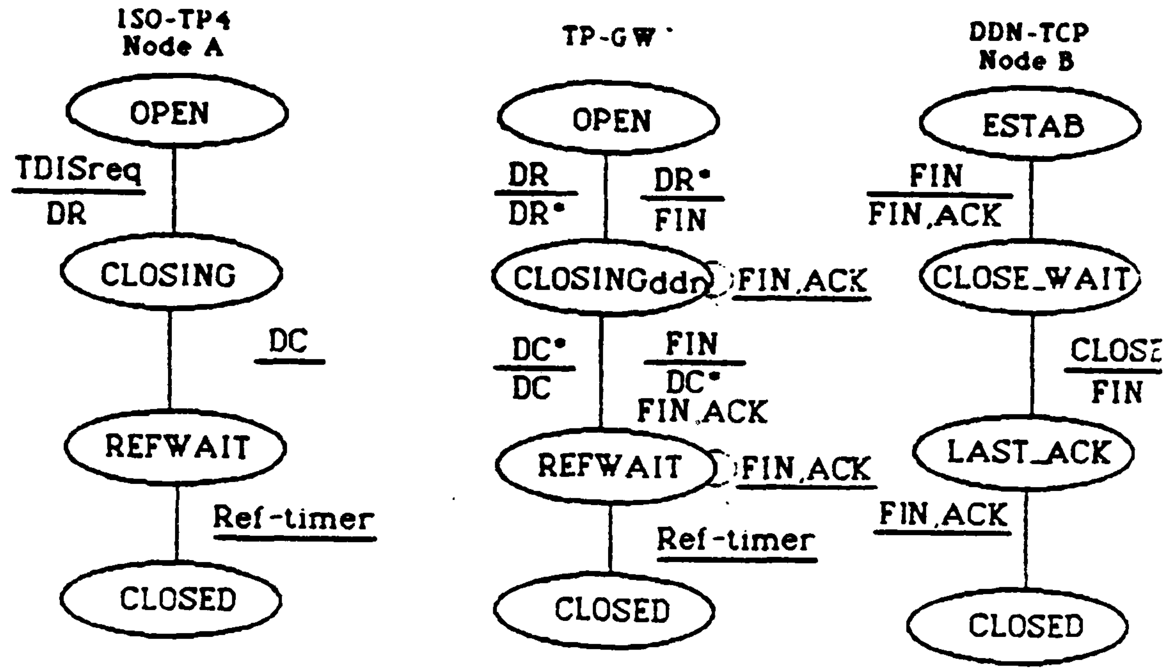


Figure 20.7 TP-Gateway Connection Release Initiated by ISO-TP4

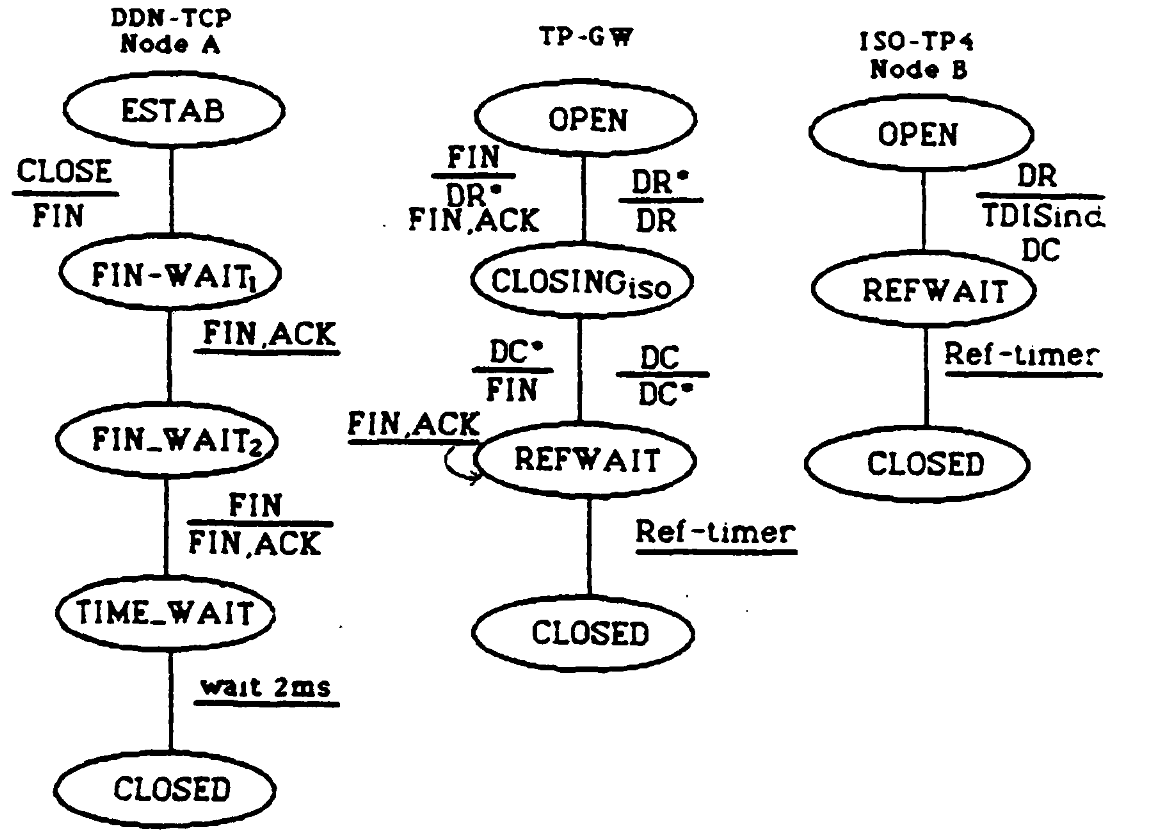


Figure 20.8 TP-Gateway Connection Release Initiated by DDN-TCP

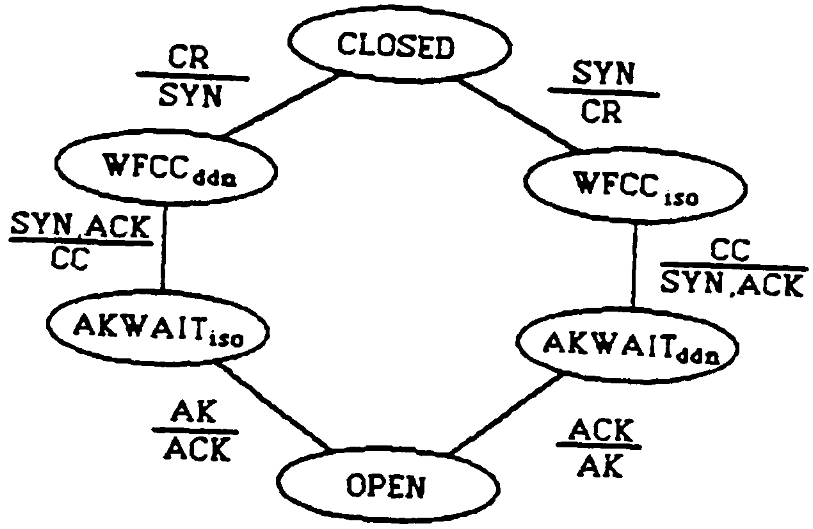


Figure 20.9 State Transition: TP-Gateway Connection Establishment

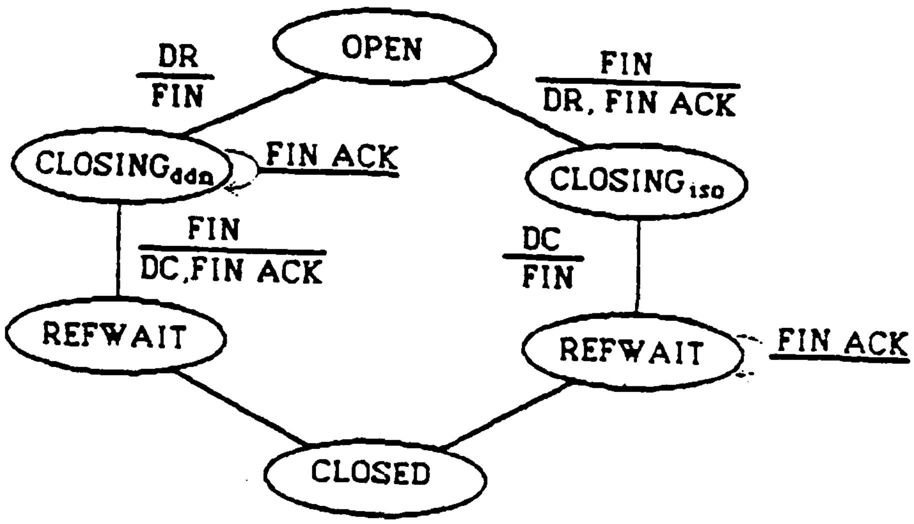


Figure 20.10 State Transition: TP-Gateway Connection Release

20.2 TRANSPORT_LAYER MODULE SPECIFICATION

Functionally speaking, this module can be further divided into two sub-modules: the TP4-dependent submodule, and the TP4-interface submodule. But due to the restrictions of the ESTELLE-like specification language, and the special situation inside the module, the relationship between the TP4-dependent and the TP4-interface submodules are specified as a parent-process and the child-processes. The TP-machines are dynamically created and terminated as the transport connection established and released. The parent process in ESTELLE can create the instance of the child process, the instances of the TP4-interface submodule are created by the TP4-dependent module upon receiving the Connection Request (CR) from ISO-IP module or from the other side of the gateway.

The commented specification for the TRANSPORT_LAYER module inside the transport gateway is enclosed in Appendix A.3.

21. CONCLUSION

Starting from the in depth discussion of internetworking requirements, this project studies the protocol interoperability issues in the DDN internet environment in detail, with emphasis on the interoperability with ISO standard protocols. Both the DDN and the ISO protocol suites are studied in terms of the basic functionalities of the layered services, service access point interfaces, and protocol peer entity mechanisms. The protocol conversions between the DDN TCP/IP and the ISO IP/TP4 is designed using ESTELLE, with the concerns of the methodology in dealing with the interoperability problem.

With such a larger volume of research efforts involved in the transport gateway design than the conventional Network Layer gateway design, it raises some very interesting questions for future studies.

1) Selection of better tools. LOTUS is very good in formal specification of protocols, and ESTELLE is a more implementation-oriented design tool. Still, some better tools are needed to overcome their shortcomings.

2) Verification of such large implementation-oriented system. With such a large system, it is very difficult to verify the design by simulation or testing. It is also painful to

convert the design to some other tools for the verification. New deveoping environment should have both the abilities for specification and verification.

3) Performance studies. For those uncertain parameters or untested strategies in such a complex system, more performance studies are needed to verify the design strategies and to improve the throughput, response time.

4) Prototype demonstrations. Because of the complexity of implementing conversion between DDN TCP/IP and ISO TP/IP networks, a prototype demonstration should be undertaken. Initially, the demonstration can be performed using a gateway between two TCP/IP and ISO TP4/IP local area networks.

It is in the interest of the USAISC to continue to research the interoperability problems between the DDN and ISO protocols. Experience gained in these efforts can yield information, specification, and prototypes which help the USAISC and support activities understanding the major issues and problems.

APPENDIX A. TRANSPORT GATEWAY SPECIFICATION

APPENDIX A.1 GLOBAL DEFINITIONS AND THE ROOT MODULE

```
specification TRANSPORT_GATEWAY;

constant

     IP_LINKER = 0;              (* IP Common Linker Module Id *)
     ISO_IP_MODULE = 1;          (* ISO-IP Module Id *)
     DoD_IP_MODULE = 2;          (* DoD-IP Module Id *)
     TP_LINKER = 8;              (* TP Common Linker Module Id *)
     ISO_TP4_MODULE = 9;         (* ISO-TP4 Module Id *)
     DoD_TCP_MODULE = 10;        (* DoD-TCP Module Id *)

     MAX_QUEUE = ...;            (* queue length limit *)


type

     octet = 0..255;        (* one byte *)
     short_word = ...;      (* two bytes *)
     word_type = ...;       (* four bytes *)
     data_type = ...;       (* uniterpreted string of bytes with
                               variable length, has semantics of
                               string pointer *)


     SNi_addr_type = ...; (* ISO SubNetwork Address *)
     SNi_QoS_type = ...;  (* ISO SubNetwork QoS *)

     SNd_addr_type = ...; (* DDN SubNetwork Address *)
     SNd_QoS_type = ...;  (* DDN SubNetwork QoS *)


     direction_type = (in,out);          (* data flow direction *)

     initiator_type = (local, remote);

     queue_type = record
          buf : array [1..MAX_QUEUE] of data_type;
          last: integer;
          end;
```

```
(***                                                    ***)
(***    Data Structure Definitions for ISO-IP PDU       ***)
(***                                                    ***)


   I_IP_addr_type = record          (* ISO-IP address format *)
        length  : 1..255;           (* length, positive *)
        addr : data_type;           (* variable length *)
        end;


   I_IP_option_type = record        (* ISO-IP option format *)
        code   : octet;             (* parameter code *)
        length : 1..255;            (* length, positive *)
        value  : data_type;         (* data, variable length *)
        end;


   I_IP_PDU_type = record              (* ISO-IP PDU format *)
        net_protocol_id : octet;       (* network_layer_protocol_id *)
        length : octet;                (* header length indicator *)
        version: octet;                (* IS8473 version, 0000 0001 *)
        lifetime: octet;               (* PDU lifetime, half-sec *)
        flags: octet;                  (* segment & errot flags *)
        seg_length: short_word;        (* PDU length *)
        checksum: short_word;          (* PDU header checksum *)
        destin_addr: I_IP_addr_type;
        source_addr: I_IP_addr_type;
                                      (* optional *)
        data_unit_id: short_word;      (* initial PDU id *)
        seg_offset: short_word;        (* data in initial PDU data *)
        total_length: short_word;      (* length of initial PDU *)
                                      (* end of optional part *)
        option: I_IP_option_type;
        data: data_type;
        end;
```

(* The PDU structures: I_IP_PDU_type here and D_IP_PDU_type below, are defined because the message exchanging between the ISO-IP module, the DDN IP module and the external world are at the system level. The values of the data fields are defined inside the ISO-IP and the DDN IP modules respectively. *)

```
(***                                                      ***)
(***                                                      ***)
(***    Data Structure Definitions for DDN-IP PDU         ***)
(***                                                      ***)

   D_IP_PDU_type = record
        I_length : octet;           (* IP version(4) & header length *)
        service : octet;            (* type of service *)
        tot_length: short_word;     (* PDU total length *)
        id: short_word;             (* PDU identification *)
        flags: octet;               (* segmentation flags *)
        offset: short_word;         (* fragment offset *)
        lifetime: octet;            (* PDU time to live, in sec *)
        protocol : octet;           (* next level protocol *)
        checksum: short_word;       (* header checksum *)
        source_addr: word_type;
        destin_addr: word_type;
        option: data_type;
        data: data_type;
        end;



(***                                                      ***)
(***    Data Structure Definitions for IP-TP              ***)
(***             Message Exchanging                       ***)
(***                                                      ***)

   IP_TP_message_type = record
        module  : octet;           (* Module Id, left 4 bits
                                          for desti. module *)
        message : octet;           (* Message Id *)
        s_IP_addr : word_type;     (* source IP address *)
        d_IP_addr : word_type;     (* destin IP address *)
        data :    data_type;       (* TPDU *)
        end;
```

(* The message format defined above are used for the communications between the Network Layer and the Transport Layer, that is, between the ISO-IP and ISO-TP submodules, and between the DoD-IP and TCP submodules. The IP_addr fields is always required to distinguish the particular connection over one pair of sockets which is made of the IP address and the Transport TSAP address. The interpretation of the data part, that is, the TPDU, is done inside the ISO-TP and DoD-TCP submodules *)

```
(***                                                         ***)
(***              Module Interaction Points                  ***)
(***                                                         ***)

  (* ISO Network Interaction Point (to the sub-network)  *)

channel I_IP_subnet_primitives (user, provider);

  by user:
     I_IP_PDUreq
        (SNi_d_addr: SNi_addr_type;
         SNi_s_addr: SNi_addr_type;
         SNi_QoS: SNi_QoS_type;
         packet : I_IP_PDU_type);

  by provider:
     I_IP_PDUind
        (SNi_d_addr: SNi_addr_type;
         SNi_s_addr: SNi_addr_type;
         SNi_QoS: SNi_QoS_type;
         packet : I_IP_PDU_type);


  (* DDN Network Interaction Point (to the sub-network) *)

channel D_IP_subnet_primitives (user, provider);

  by user:
     D_IP_PDUreq
        (SNd_d_addr: SNd_addr_type;
         SNd_s_addr: SNd_addr_type;
         SNd_QoS: SNd_QoS_type;
         packet : D_IP_PDU_type);

  by provider:
     D_IP_PDUind
        (SNd_d_addr: SNd_addr_type;
         SNd_s_addr: SNd_addr_type;
         SNd_QoS: SNd_QoS_type;
         packet : D_IP_PDU_type);
```

```
(* Network Service Interaction Point (to Transport Layer) *)

channel NCEP_primitives (user, provider);

  by user:
     IP_TP_MESreq (message : IP_TP_message_type);

  by provider:
     IP_TP_MESind (message : IP_TP_message_type);




  (***                                                     ***)
  (***                 Module Definitions                  ***)
  (***                                                     ***)


module NETWORK_LAYER_entity_type process
     (SNSi: I_IP_subnet_primitives (user);
     (SNSd: D_IP_subnet_primitives (user);
      NSi:  NCEP_primitives (provider);
      NSd:  NCEP_primitives (provider);
     );



module TRANSPORT_LAYER_entity_type process
     (NSi:  NCEP_primitives (user);
      NSd:  NCEP_primitives (user);
     );



body NETWORK_LAYER_entity_body for NETWORK_LAYER_entity_type;
                                                       external;

body TRANSPORT_LAYER_entity_body for TRANSPORT_LAYER_entity_type;
                                                       external;
```

```
(***                                                          ***)
(***              Common Procedures                           ***)
(***                                                          ***)


function d_length (mes: data_type) : integer; primitive;

     (* calculate the length of the string *)


procedure d_append (var mes: data_type;
                        mes2: data_type); primitive;

     (* append string mes2 to mes *)


function d_create (l: integer) : data_type; primitive;

     (* allocate data string space, initialize to zeros *)



function d_get (pdu: data_type; offset: integer) : octet;

     (* get one byte from the string *)

     begin
     d_get := pdu[offset];
     end;



function d_gets (pdu: data_type;
                 offset,l: integer) : data_type;

     (* get a sub-string of length l from a string at offset *)

  var
     mes: data_type;
     i: integer;

     begin
     mes := d_create(l);
     for i := 1 to l do
          mes[i] := d_get(pdu, offset+i-1);
     d_gets := mes;
     end;
```

```
function get_sword (mes: data_type; l:integer) : short_word;

     (* get a short_word of length l from the string *)

  var
     sw: short_word;
     i: integer;

     begin
     sw := 0;
     for i := 1 to l do
          sw := sw*256 + mes[i];
     get_sword := sw;
     end;



function get_word (mes: data_type; l:integer) : word_type;

     (* get a word of length l from the string *)

  var
     w: word_type;
     i: integer;

     begin
     w := 0;
     for i := 1 to l do
          w := w*256 + mes[i];
     get_word := w;
     end;



procedure d_put (var pdu: data_type;
                    offset: integer; byte: octet);

     (* put one byte into the string *)

     begin
     pdu[offset] := byte;
     end;
```

```
procedure d_puts (var pdu: data_type;
                  offset: integer; segment: data_type);

     (* put s sub-string into string at offset *)

  var
     l,i: integer;

     begin
     l := d_length(segment);
     for i := 1 to l do
          pdu[offset+i-1] := segment[i];
     end;



procedure d_bitset (var byte: octet;
                    bit: integer;
                    flag: boolean) : octet;

     (* set the bit, as arranged to 8..1 *)

  var
     b: octet;

     begin
     b := 1;
     for bit := bit-1 to 1 do
          b := b*2;                       (* shift left *)
     if (flag) then
          byte := byte or b;              (* set *)
       else
          byte := byte and (not b);       (* reset *)
     end;
```

```
function d_encode (n: integer) : octet;

    (* convert integer to octet (data_type) *)

  var
    byte: octet;
    i: integer;

    begin
    byte := 0;
    for i := 1 to 8 do
         begin
         if (n mod 2<>0) then
              d_bitset (byte,i,true);
         n := n div 2;
         end;
    d_encode := byte;
    end;



function d_encode2 (n: integer) : data_type;

    (* encode integer to 2-byte string *)

  var
    mes: data_type;

    begin
    mes := d_create(2);
    d_put(mes,1, n div 256);
    d_put(mes,2, n mod 256);
    d_encode2 := mes;
    end;



function d_encode4 (n: integer); data_type;

    (* encode integer to 4-byte string *)

  var
    mes: data_type;

    begin
    mes := d_create(4);
    d_puts(mes,1, d_encode2(n div 65536));
    d_puts(mes,3, d_encode2(n mod 65536));
    d_encode4 := mes;
    end;
```

```
procedure clqueue (var Q: queue_type);

     (* clear queue Q *)

     begin
     Q.last := 0;
     end;



procedure enqueue (var Q: queue_type;
                       elem: data_type);

     (* put elem in queue Q at the tail *)

     begin
     if (Q.last < MAX_QUEUE) then
       with Q do
          begin
          last := last+1;
          Q[last] := elem;
          end;
     end;



function dequeue (var Q: queue_type) : data_type;

     (* get head of FIFO queue *)

     var  i: integer;

     begin
     if (Q.last > 0) then
       with Q do
          begin
          dequeue := buf[1];
          for i := 1 to last-1 do
               buf[i] := buf[i+1];
          last := last-1;
          end;
      else
          dequeue := d_null;
     end;
```

```
function exqueue (var Q: queue_type;
                      i: integer) : data_type;

     (* get ith element of queue *)

     var  j: integer;

     begin
     if (Q.last >= i) then
       with Q do
          begin
          exqueue := buf[i];
          for j := i to last-1 do
               buf[j] := buf[j+1];
          last := last-1;
          end;
      else
          exqueue := d_null;
     end;



procedure requeue (var Q: queue_type;
                       i: integer);

     (* renew the elem in queue Q by putting it at the tail *)

     var  j: integer;
          elem: data_type;

     begin

     if (Q.last >= i) then
       with Q do
          begin
          elem := Q[i];
          for j := i to last-1 do
               buf[j] := buf[j+1];
          Q[last] := elem;
          end;
     end;
```

```
function chk_checksum (pdu: data_type;
                       leng: integer) : boolean;

     (* check the PDU_header checksum *)

  var
     C0,C1,i : integer;

     begin
     C0 := 0;
     C1 := 0;
     for i := 1 to leng do
          begin
          C0 := (C0 + d_get(pdu,i)) mod MODULUS;
          C1 := (C1 + C0) mod MODULUS;
          end;
     if ((C0=0) and (C1=0)) then
          checksum := true
      else
          checksum := false;
     end;



procedure set_checksum (var pdu: data_type;
                        length, CS: integer);

     (*  calculate  the PDU checksum,
          length is the length of pdu header,
          CS is the position of CheckSum *)

  var
     X,Y : octet;
     C0,C1,i : integer;

     begin
     C0 := 0;
     C1 := 0;
     for i := 1 to length do
          begin
          C0 := (C0 + d_get(pdu,i)) mod MODULUS;
          C1 := (C1 + C0) mod MODULUS;
          end;
     length := length - CS;
     X := (-C1 + length*C0) mod MODULUS;
     Y := (C1-(length+1)*C0) mod MODULUS;
     d_put(pdu, CS, X);
     d_put(pdu, CS+1, Y);
     end;
```

```
(***                                          ***)
(***          initialization                  ***)
(***                                          ***)


initialize

     init NETWORK_LAYER with NETWORK_LAYER_entity_body();
     connect SNSi to NETWORK_LAYER.SNSi (user);
     connect SNSd to NETWORK_LAYER.SNSd (user);
     connect NSi to NETWORK_LAYER.NSi (provider);
     connect NSd to NETWORK_LAYER.NSd (provider);


     init TRANSPORT_LAYER with TRANSPORT_LAYER_entity_body();
     connect NSi to TRANSPORT_LAYER.NSi (user);
     connect NSd to TRANSPORT_LAYER.NSd (user);

end;

(* The end of the specification.  It is a container with common
data types, common procedures and functions, module
declarations, and the initialization.  The rest modules are
defined externally.  *)
```

APPENDIX A.2 NETWORK_LAYER MODULE SPECIFICATION

```
(***                                                          ***)
(***          NETWORK_LAYER Module Specification             ***)
(***                                                          ***)


body NETWORK_LAYER_entity_body for NETWORK_LAYER_entity_type;


  constant

     SOURCE_ROUTING = 1100 1000; (* opt. code for source routing *)
     RECORD_ROUTING = 1100 1011; (*                record route *)
     ERROR_REASON = 1100 0001;   (*                error reason *)
     SECURITY = 1100 0101;       (*                security *)
     QOS = 1100 0011;            (*                QoS *)
     PRIORITY = 1100 1101;       (*                Priority *)
     PADDING = 1100 1100;        (*                padding *)

     GW_TRANS_DELAY = ...;       (* estimated GW transit delay *)


  type

     (* Types of Reasons for Error_report PDUs *)

     error_type  = (NO_ERROR,
                    NOT_SPECIFIED,
                    PROCOTOL_PROC_ERROR,
                    INCORRECT_CHECKSUM,
                    CONGESTION,
                    HEADER_SYNTAX_ERROR,
                    DATA_TOO_LONG,
                    D_ADDR_UNREACHABLE,
                    D_ADDR_UNKNOWN,
                    S_ROUTING_UNSPECIFIED,
                    S_ROUTING_SYNTAX_ERROR,
                    S_ROUTING_UNKNOWN_ADDR,
                    PATH_NOT_ACCEPTABLE,
                    LIFETIME_EXPIRED_IN_TRANSIT,
                    LIFETIME_EXPIRED_IN_REASSEMBLY);


     (* Address_field format, as the components of route option *)

     addr_field_type = record
          length: octet;           (* length of addr field *)
          addr: data_type;         (* addr field *)
          end;
```

```
(* Routing Option format *)

routing_option_type = record
     code: octet;                    (* source/record routing *)
     length: octet;                  (* length of the option *)
     flag: octet;                    (* s_routing: 1=complete;
                                                   0=partial;
                                        r_route:   0=in progress;
                                               1111 1111=terminated *)
     ptr: octet;                     (* ptr to addr fields, from 3 *)
     fields: addr_field_type;
     end;


(***    Data Structure Definitions for the Message    ***)
(***    Exchanging between IP Dependent Submodules    ***)
(***            and Common Linker Module              ***)


IP_message_type = record        (* IP message format *)
     module      : octet;       (* Module Id *)
     message     : octet;       (* message Id *)
     source_addr: word_type;    (* limited to 4 bytes *)
     destin_addr: word_type;    (* limited to 4 bytes *)
     lifetime    : octet;       (* PDU lifetime, half-sec *)
     option      : I_IP_option_type;
     in_time     : octet;       (* time arriving in GW *)
     end;

(* The message format defined above is used to carry information
between the IP modules with "the common linker module".  It is
necessary to overcome the problem that the TPDU data does not
always carry the network addresses and some other network control
information such as lifetime of PDU which are necessary for
network layer control.  The information will be picked up at the
other side of gateway to assemble the network PDU header before
sending the PDU to the remote network. *)


(***         Data  Structure  Definitions             ***)
(***      for Storing Control Information of          ***)
(***               a IP_message                       ***)


IP_message_handler_type = record
     message     : octet;       (* message Id *)
     lifetime    : octet;       (* PDU lifetime, half-sec *)
     option      : I_IP_option_type;
     in_time     : octet;       (* keep transit time *)
     end;
```

```
(***          Data  Structure  Definitions              ***)
(***       for Storing Control Information of           ***)
(***               a Pair IP Nodes                      ***)


   IP_segment_type = record        (* segment & time_marker *)
        pdu: data_type;            (* PDU segment *)
        time: octet;               (* time marker *)
        end;


   IP_pair_type = record
        module: octet;             (* module id *)
        s_IP_addr: word_type;      (* source IP address *)
        d_IP_addr: word_type;      (* destin IP address *)
        s_routing: routing_option_type;
                                   (* source routing to destin *)
        r_routing: routing_option_type;
                                   (* record routing to source *)
        unitid: short_word;        (* unit_id for sending *)
        mes_id: octet;             (* mes_id for sending *)
        rcv_mes_que: queue_type of IP_message_handler_type;
                                   (* receive message id queue *)
        send_mes_que: queue_type of IP_message_handler_type;
                                   (* send message id queue *)
        segment_que: queue_type of IP_segment_type;
                                   (* segmentation queue *)
        timestamp: word_type;      (* timestamp of action *)
        end;



(***                                                    ***)
(***            Module Interaction Points               ***)
(***                                                    ***)


 (* IP Network Gateway Interaction Point (to linker module) *)

channel IP_linker_primitives (user, provider);

  by user:
     IP_MESreq (message : IP_message_message_type);

  by provider:
     IP_MESind (message : IP_message_message_type);
```

```
(***                                                      ***)
(***              Submodule Definitions                   ***)
(***                                                      ***)


module ISO_IP_entity_type process
     (NET_INi: I_IP_subnet_primitives (user);
      NET_GWi: IP_linker_primitives (provider);
      NET_OUTi:  NCEP_primitives (provider);
     );
   export
     i_module_id: integer;      (* ISO IP module id *)
     isn_rcved: integer;        (* packets received from ISO SN *)
     isn_sent: integer;         (* packets sent to ISO SN *)
     int_sent: integer;         (* packets sent to ISO TP *)
     int_rcved: integer;        (* packets received from ISO TP *)
     isn_error: integer;        (* errors in packets with ISO SN *)
     int_error: integer;        (* errors in packets with ISO TP *)
     end;


module DDN_IP_entity_type process
     (NET_INd: D_IP_subnet_primitives (user);
      NET_GWd: IP_linker_primitives (provider);
      NET_OUTd:  NCEP_primitives (provider);
     );
   export
     d_module_id: integer;      (* DDN IP module id *)
     dsn_rcved: integer;        (* packets received from DDN SN *)
     dsn_sent: integer;         (* packets sent to DDN SN *)
     dnt_sent: integer;         (* packets sent to DDN TCP *)
     dnt_rcved: integer;        (* packets received from DDN TCP *)
     dsn_error: integer;        (* errors in packets with DDN SN *)
     dnt_error: integer;        (* errors in packets with DDN TCP *)
     end;


body ISO_IP_entity_body for ISO_IP_entity_type; external;

body DDN_IP_entity_body for DDN_IP_entity_type; external;
```

```
(***                                       ***)
(***         initialization                ***)
(***                                       ***)

initialize

    init ISO_IP with ISO_IP_entity_body();
    connect SNSi to ISO_IP.NET_INi;          (* with ISO SubNet *)
    connect NSi to ISO_IP.NET_OUTi;          (* with ISO TP-4 *)
    connect NET_GWi to ISO_IP.NET_GW;
    ISO_IP.i_module_id := 1;
    ISO_IP.isn_rcved := 0;
    ISO_IP.isn_sent := 0;
    ISO_IP.int_sent := 0;
    ISO_IP.int_rcved := 0;
    ISO_IP.isn_error := 0;
    ISO_IP.int_error := 0;

    init DDN_IP with DDN_IP_entity_body();
    connect SNSd to DDN_IP.NET_INd;          (* with DDN SubNet *)
    connect NSd to DDN_IP.NET_OUTd;          (* with DDN TCP *)
    connect NET_GWd to DDN_IP.NET_GW;
    DDN_IP.d_module_id := 2;
    DDN_IP.dsn_rcved := 0;
    DDN_IP.dsn_sent := 0;
    DDN_IP.dnt_sent := 0;
    DDN_IP.dnt_rcved := 0;
    DDN_IP.dsn_error := 0;
    DDN_IP.dnt_error := 0;
    end;


(***                                       ***)
(***        State Transition               ***)
(***                                       ***)

trans
  when NET_GWi.IP_MESind      (* from ISO_IP to Common Linker *)

    begin
    output NET_GWd.IP_MESreq(packet);  (* send to DDN_IP *)
    end;


trans
  when NET_GWd.IP_MESind      (* from DDN_IP to Common Linker *)

    begin
    output NET_GWi.IP_MESreq(packet);  (* send to ISO_IP *)
    end;

end;      (* The End of NETWORK_LAYER module (Common Linker) *)
```

APPENDIX A.2.1 ISO_IP SUBMODULE

```
(***                                                    ***)
(***          ISO_IP Submodule Specification            ***)
(***                                                    ***)

body ISO_IP_entity_body for ISO_IP_entity_type;

  constant

     I_IP_MAX_SEGMENT_LENGTH = ...;
     I_NET_DELAY  = ...;             (* ISO Subnet transmit delay *)
     I_REASSEMBLY_DELAY = ...;       (* Reassembly delay up_limit *)
     I_LIFETIME := ...;              (* Lifetime for PDUs *)
     I_IP_ADDRESS := ...;            (* local IP address *)

     I_SN_LOCAL_ADDR := ...;         (* local SubNet address *)
     I_SN_QOS := ...;                (* local SubNet QoS *)

     I_NET_PROTOCOL_ID = 129;        (* 1000 0001 in binary *)
     I_IP_VERSION = 1;               (* 0000 0001 *)

     SEGMENT_PERMIT = 128;           (* 1000 0000 *)
     MORE_SEGMENT = 64;              (* 0100 0000 *)
     ERROR_REPORT = 32;              (* 0010 0000 *)

     DATA_PDU = 28;                  (* 0001 1100 *)
     ERROR_PDU = 1;                  (* 0000 0001 *)

     FLAG_type = (OK, EXPIRED, INCOMPLETE);


  var
     I_IP_pair: queue_type of IP_pair_type;
     mod_id: octet;
     mes_id: integer;
     cur_I_IP_pdu,err_pdu: I_IP_PDU_type;
     cur_IP_TP_mes: IP_TP_message_type;
     cur_IP_mes: IP_message_type;
     cur_IP_handler: IP_message_handler_type;
     opt_ptr: I_IP_option_type;
     s_ip_addr,d_ip_addr: word_type;
     s_route,r_route: routing_option_type;
     I_SN_addr: SNi_addr_type;

     pair_index: integer;
     err_flag: error_type;
     transit_time: octet;
     pdu_error: integer;
```

```
(***                                                     ***)
(***              Common Procedures                      ***)
(***                                                     ***)


function get_data_unit_id : short_word;

          (* get unique Data_Unit_Id *)

     begin
     with I_IP_pair[pair.index] do
          begin
          unitid := unitid+1;
          get_data_unit_id := unitid;
          end;
     end;


function get_option_length(pdu: I_IP_PDU_type) : integer;

          (* get the length of options *)

     begin
     get_option_length := pdu.length - pdu.destin_addr.length
                                   - 11 - pdu.source_addr.length;
     if (pdu.flags and SEGMENT_PERMIT) then
          get_option_length := get_option_length - 6;
     end;


function conv_routing(opt_ptr: I_IP_option_type) :
                                        routing_option_type;

          (* convert the routing direction *)

     var
          i,j,l,k,m: integer;
          opl: ^routing_option_type;

     begin
     m := opt_ptr.value[2]-1;         (* use the ptr as length *)
     opl := d_create(m+2);
     opl.code := SOURCE_ROUTING;
     opl.length := m;
     opl.flag := 1;               (* type 1 as complete routing *)
     opl.ptr := 3;                  (* ptr to beginning *)

     i := 3;                        (* copied from beginning *)
     j := m-1;                      (* copied to from end:3-1 *)
     while (i<m) do                 (* reverse the order *)
          begin
          l := opt_ptr.value[i];    (* length of field *)
```

```
            j := j - 1 - 1;
            op1.fields[j] := 1;
            for k := 1 to 1 do
                op1.fields[j+k] := opt_ptr.value[i+k];
            end;
        conv_routing := op1;
        end;



    procedure one_segment(pdu: I_IP_PDU_type);

            (* output one IP_PDU *)

        var
            i,j,k,leng: integer;
            s: data_type;

        begin
        with pdu do
            begin
            net_protocol_id := I_NET_PROTOCOL_ID;
            version := I_IP_VERSION;
            seg_length := length + d_length(data);

            leng := get_option_length(pdu);
            opt_ptr := ^option;
            s_route := d_null;    (* looking for SOURCE_ROUTING *)
            while ((s_route=d_null) and (leng>0)) then
              if (opt_ptr.code=SOURCE_ROUTING) then
                 s_route:=d_gets(opt_ptr,1,opt_ptr.length+2);
               else
                 begin
                 leng := leng - opt_ptr.length;
                 opt_ptr := ^opt_ptr.value[opt_ptr.length+1];
                 end;
            if (s_route=d_null) then
                 s_route :=
                   conv_routing(I_IP_pair.buf[pair_index].r_routing);
            k := s_route.ptr - 2;     (* pointer to field *)
            j := s_route.fields[k];  (* the length *)
            s := d_gets(s_route.fields, k+1, j);      (* get field *)
            I_SN_addr := trans_SN_addr(s); (* translate to SN_addr *)
            opt_ptr.value[2] := opt_ptr.value[2] + j + 1;
                                          (* modify the pointer *)
            checksum := 0;
            set_checksum (pdu, length, 7);
            end;

        output NET_INi.I_IP_PDUreq (I_SN_addr,
                                    I_SN_LOCAL_ADDR,
                                    I_SN_QOS,
                                    pdu);
        end;
```

```
procedure error_pdu (err: error_type;
                     ptr,leng: integer;
                     pdu: I_IP_PDU_type);

          (* send an Error_Report pdu *)

     var
          opt_ptr: I_IP_option_type;

     begin
     opt_ptr := d_create(4);
     opt_ptr.code := ERROR_REASON;
     opt_ptr.length := 2;
     opt_ptr.value[1] := err;
     opt_ptr.value[2] := ptr;
     d_append(opt_ptr,I_IP_pair.buf[pair_index].r_routing);

     with err_pdu do                   (* Make ERROR PDU *)
          begin
          lifetime := I_LIFETIME;
          flags := (pdu.flags and SEGMENT_PERMIT) or ERROR_PDU;
          destin_addr.length := pdu.source_addr.length;
          destin_addr.addr := pdu.source_addr.addr;
          source_addr.length := 4;
          source_addr.addr := I_IP_ADDRESS;
          option := opt_ptr;
          length := 11 + destin_addr.length + source_addr.length
                       + d_length(option);
          data := d_gets(pdu, 1, leng);
          if (flags and SEGMENT_PEMIT) then
               begin
               length := length + 6;
               data_unit_id := get_data_unit_id();
               seg_offset := 0;
               total_length := length + d_length(data);
               end;
          one_segment(err_pdu);
          end;
     end;


function pair_match : integer;

          (* match the I_IP_pair with s/d_ip_addr,
             set and return the index *)

     var
          i,j,k,leng: integer;
          t: word_type;

     begin
     j := 0;
```

```
for i := 1 to I_IP_pair.last do    (* look for the pair with
                                      same IP_address info *)
  with I_IP_pair.buf[i] do
     if ((s_IP_addr = s_ip_addr) and
         (d_IP_addr = d_ip_addr)) then
          j := i;

 if ((j<>0) and (mod_id=0)) then                (* matched *)
     mod_id := I_IP_pair.buf[j].module/16;
 else if ((j<>0) and (mod_id<>0) and        (* matched *)
          (I_IP_pair.buf[j].module/16=0)) then
     I_IP_pair.buf[j].module := mod_id*16 + i_module_id;
 else if (j=0) then    (* need fill in new IP_pair entry *)
     begin
     if (I_IP_pair.last<MAX_QUEUE) then
          begin
          I_IP_pair.last := I_IP_pair.last+1;
          j := I_IP_pair.last;
          end;
      else                                     (* no empty slot *)
          begin
          t := cur_time();
          for i := 1 to I_IP_pair.last do
            if (I_IP_pair.buf[i].timestamp<t) then
               begin
               j := i;
               t := I_IP_pair.buf[i].timestamp;
               end;

          pair_index := j;
          for k := 1 to segment_que.last do
                    (* clean reassembly queue by Error/Report *)
            with segment_que.buf[k] do
               error_pdu(CONGESTION, 0, pdu.length, pdu);
                                     (* send Error/Report *)
          end;

     with I_IP_pair.buf[j] do (* fill in *)
          begin               (* use empty/oldest slot *)
          module = mod_id * 16 + i_module_id;
          s_IP_addr := s_ip_addr;
          d_IP_addr := d_ip_addr;
          s_routing := d_null;
          d_routing := d_null;
          rcv_mes_que.last := 0;
          send_mes_que.last := 0;
          segment_que.last := 0;
          timestamp = cur_time();
          end; (* with: fill in *)
     end;  (* if j=0 *)

pair_index := j;
pair_match := j;
end;
```

```
(***                                              ***)
(***            initialization                    ***)
(***                                              ***)


initialize

     pdu_error := 0;
     I_IP_pair.last := 0;
end;




(***                                              ***)
(***          State Transition                    ***)
(***                                              ***)


trans
  when NET_INi.I_IP_PDUind              (* from ISO subnetwork *)

     var
          i,ii,j,k,leng,total: integer;
          t: word_type;
          pdu1: I_IP_PDU_type;
          flag: FLAG_type;

    begin
    transit_time := cur_time();
    cur_I_IP_pdu := packet;
    if (cur_I_IP_pdu.source_addr.length>4) then
          pdu_error := pdu_error+1;     (* PDU dropped *)
      else if (cur_I_IP_pdu.destin_addr.length>4) then
          pdu_error := pdu_error+1;     (* PDU dropped *)
      else
        begin
        s_ip_addr := cur_I_IP_pdu.source_addr.addr;
        d_ip_addr := cur_I_IP_pdu.destin_addr.addr;
        mod_id := 0;
        j := pair_match();
        err_flag := NO_ERROR;
        s_route := d_null;
        r_route := d_null;
        with cur_I_IP_pdu do
          with I_IP_pair.buf[j] do
            begin
            if(chk_checksum(cur_I_IP_pdu,length)=false) then
                    err_flag := INCORRECT_CHECKSUM;
             else
               begin
```

```
lifetime := lifetime - I_LIFETIME;
leng := get_option_length(cur_I_IP_pdu);
opt_ptr := ^option;
while (leng>0) then
     if (opt_ptr.code=SOURCE_ROUTING) then
          begin
          s_route:=d_gets(opt_ptr,1,opt_ptr.length+2);
          s_routing := s_route;
          end;
      else if (opt_ptr.code=RECORD_ROUTING) then
          begin
          r_route := conv_routing(opt_ptr);
          r_routing := r_route;
          end;
      else
          begin
          leng := leng - opt_ptr.length;
          opt_ptr := ^opt_ptr.value[opt_ptr.length+1];
          end;
 end;
if((err_flag=NO_ERROR) and (lifetime<=0)) then
    err_flag := LIFETIME_EXPIRED_IN_TRANSIT;
 else if((err_flag=NO_ERROR) and (s_route=d_null)) then
    err_flag := S_ROUTING_UNSPECIFIED;

if (err_flag<>NO_ERROR) then  (* send Error/Report *)
   begin
   error_pdu(err_flag, 1, length, cur_I_IP_pdu);
   pdu_error := pdu_error+1;     (* PDU dropped *)
   end;
 else
   begin                          (* handle segment  *)
   flag := OK;
   ii := 0;
   if (flags and SEGMENT_PERMIT=0) then
        begin
        pdu1 := cur_I_IP_pdu;    (* no segmentation *)
        total := seg_length;
        k := total;
        end;
    else
        begin
        total := total_length;
        i := 1;
        while ((i<segment_que.last) and (data_unit_id
              <segment_que.buf[i].pdu.data_unit_id)) do
              i := i + 1;
        ii := i;
        k := 0;
        while ((flag=OK)and (i=<segment_que.last) and
          (k<total) and (data_unit_id
              =segment_que.buf[i].pdu.data_unit_id)) do
          with segment_que.buf[i] do
             begin
             if (seg_offset=k) then (* of cur_I_IP_pdu *)
```

```
                    begin
                    if (k=0) then
                         pdu1 := cur_I_IP_pdu;
                     else
                         d_append(pdu1.data, data);
                    k := k+seg_length-length;
                                   (* Length of data *)
                    end;
               if (pdu.lifetime<=cur_time()- time) then
                    flag := EXPIRED;
                else if (pdu.seg_offset>k) then
                    flag := INCOMPLETE;
                else
                    begin
                    if (k=0) then
                         pdu1 := segment_que.buf[i];
                     else
                         d_append(pdu1.data,pdu.data;
                    k := k+pdu.seg_length-pdu.length;
                    end;
                    if (flag=OK) then
                         i := i+1;
                     else                (* stop *)
                         i := segment_que.last+1;
               end;  (* while *)
          i := ii;
          end;  (* else SP *)
     if (((flag=OK) and (k<total)) or  (* miss last seg.*)
         (flag=INCOMPLETE) then
          begin  (* PDU is not complete yet, queueed *)
          for k := i to segment_que.last do
               segment_que.buf[i+1] := segment_que.buf[i];
          segment_que.buf[i].pdu := cur_I_IP_pdu;
          segment_que.buf[i].time := cur_time();
          segment_que.last := segment_que.last+1;
          end;
      else if (i>0) then (* need to extract segments
                              expired/reassemblied *)
          while ((i<segment_que.last) and (data_unit_id=
                segment_que.buf[i].pdu.data_unit_id)) do
               begin
               exqueue(segment_que, i);
               i := i+1;
               end;

     if (flag=EXPIRED) then
          error_pdu(LIFETIME_EXPIRED_IN_REASSEMBLY,
                    1, length, cur_I_IP_pdu);
      else if(k=total) then
          begin
          with cur_IP_TP_mes do
               begin
               module := I_IP_pair.buf[j].module;
               mes_id := mes_id + 1;
               if (mes_id=0) then
```

```
                                     mes_id := 1;
                                message := mes_id;
                                s_IP_addr := s_IP_addr;
                                d_IP_addr := d_IP_addr;
                                data := pdu1.data;
                                end;
                           output NET_OUTi.I_IP_PDUind (cur_IP_TP_mes);

                           with cur_IP_mes do
                                begin          (* make GW packet *)
                                module := I_IP_pair.module;
                                message := mes_id;
                                source_addr := s_IP_addr;
                                destin_addr := d_IP_addr;
                                lifetime := pdu1.lifetime
                                            - (cur_time()-transit_time);
                                option := pdu1.option;
                                end;
                           output NET_GWi.IP_MESind (cur_IP_mes);

                           with cur_IP_handler do
                                begin     (* make IP_message_handler *)
                                message := mes_id;
                                lifetime := pdu.lifetime;
                                option := pdu.option;
                                end;
                           enqueue (rcv_mes_que, cur_IP_handler);

                           end;  (* if flag *)
                      end;  (* else err_flag *)

                timestamp := cur_time;
                end;  (* with-with *)
            end;  (* else *)
        end;
```

```
trans
  when NET_GWi.IP_MESreq             (* from GW common linker *)

     var
          j,leng: integer;

     begin
     cur_IP_mes := packet;
     with cur_IP_mes do
          begin
          mod_id := module/16 + (module mod 16)*16;
          d_ip_addr := source_addr;
          s_ip_addr := destin_addr;
          j := pair_match();

          with I_IP_pair.buf[j] do
               begin
               s_route := s_routing;
               r_route := r_routing;
               if ((s_route=d_null) or (r_route=d_null)) then
                    begin
                    opt_ptr := ^option;
                    leng := d_length(option);
                    while (leng>0) then
                      if ((r_route=d_null) and
                           (opt_ptr.code=SOURCE_ROUTING)) then
                         begin
                         r_route:= conv_routing(opt_ptr);
                         r_routing := r_route;
                         end;
                     else if ((s_route=d_null) and
                               (opt_ptr.code=RECORD_ROUTING)) then
                         begin
                         s_route := d_gets(opt_ptr,1,opt_ptr.length+2);
                         s_routing := s_route;
                         end;
                     else
                         begin
                         leng := leng - opt_ptr.length;
                         opt_ptr := ^opt_ptr.value[opt_ptr.length+1];
                         end;
                end; (* if *)
               with cur_IP_handler do
                    begin      (* make IP_message_handler *)
                    message := mes_id;
                    lifetime := pdu.lifetime;
                    option := pdu.option;
                    in_time := cur_time();
                    end;
               enqueue (send_mes_que, cur_IP_handler);
               timestamp := cur_time;
               end;  (* with I_IP_pair.buf[j] *)
          end;  (* with cur_IP_mes *)
     end;
```

```
trans
  when NET_OUTi.I_IP_PDUreq                        (* from TP-4 *)

      var
           i,j,k,n: integer;
           handler: IP_message_handler_type;
           ip_data: data_type;

      begin
      cur_IP_TP_mes := packet;
      with cur_IP_TP_mes do
           begin
           mod_id := module/16 + (module mod 16)*16;
           mes_id := message;
           s_ip_addr := d_IP_addr;
           d_ip_addr := s_IP_addr;

           j := 0;
           for i := 1 to I_IP_pair.last do
             with I_IP_pair.buf[i] do
                if ((s_ip_addr=s_IP_addr) and
                    (d_ip_addr=d_IP_addr)) then
                          j := i;

           if (j=0) then
                pdu_error := pdu_error+1;
                     (* It is supposed to have a match here
                        because even if the IP_module at the
                        gateway side initiated the session, the
                        IP_message has been transferred over
                        faster to establish an entry earlier. *)
            else
              with I_IP_pair.buf[j] do
                begin
                pair_index := j;                  (* global *)
                k := 0;
                if (mes_id<>0) then
                  for i := 1 to send_mes_que.last do
                    if (send_mes_que.buf[i].message=mes_id) then
                      k := i;
                if (k=0) then
                     k := 1;       (* if no match, use 1st one *)
                handler := exqueue(send_mes_que, k);
                with cur_I_IP_pdu do
                     begin
                     destin_addr.length := 4;
                     destin_addr.addr := s_ip_addr;
                     source_addr.length := 4;
                     source_addr.addr := d_ip_addr;
                     flags := ERROR_REPORT or DATA_PDU;
                     option := handler.option;
                     lifetime := handler.lifetime
                                - (cur_time() - handler.in_time);
                     length := d_length(option) + 19;
```

```
                data := cur_IP_TP_mes.data;
                                    (* segmentation & output *)
                n := d_length(data);
                if (n > I_IP_MAX_SEGMENT_LENGTH) then
                     begin
                     length := length+6;
                     flags := flags or SEGMENT_PERMIT;
                     data_unit_id := get_data_unit_id;
                     seg_offset := 0;
                     total_length := length + n;
                     ip_data := data;
                     i := 1;
                     while (n>i*I_IP_MAX_SEGMENT_LENGTH) do
                          begin
                          flags := flags or MORE_SEGMENT;
                          seg_offset := seg_offset +
                                        I_IP_MAX_SEGMENT_LENGTH;
                          data := d_gets(ip_data,
                                  (i-1)*I_IP_MAX_SEGMENT_LENGTH+1
                                  I_IP_MAX_SEGMENT_LENGTH);
                          one_segment(cur_I_IP_pdu);
                          i := i+1;
                          n := d_length(ip_data);
                          end;
                     flags := flags and (not MORE_SEGMENT);
                     data:=d_gets(ip_data,
                           (i-1)*I_IP_MAX_SEGMENT_LENGTH+1,
                           d_length(ip_data)
                                  -I_IP_MAX_SEGMENT_LENGTH);
                     end;  (* if n > *)

                one_segment();
                end;  (* with cur_I_IP_pdu *)
           timestamp := cur_time;
           end;  (* with I_IP_pair.buf[j] *)
      end;  (* with cur_IP_TP_mes *)
 end;  (* trans *)

end;    (* of ISO_IP_entity_body *)
```

APPENDIX A.2.2 DDN_IP SUBMODULE

```
(***                                                    ***)
(***         DDN_IP Submodule Specification             ***)
(***                                                    ***)

body DDN_IP_entity_body for DDN_IP_entity_type;

  constant

     D_IP_MAX_SEGMENT_LENGTH = ...;
     D_NET_DELAY  = ...;              (* DDN Subnet transmit delay *)
     D_REASSEMBLY_DELAY = ...;        (* Reassembly delay up_limit *)
     D_LIFETIME := ...;               (* Lifetime for PDUs *)
     D_IP_HOST_ADDRESS := ...;        (* local IP host address *)
     D_IP_NET_ADDRESS := ...;         (* local IP net address *)

     D_SN_LOCAL_ADDR := ...;          (* local SubNet address *)
     D_SN_QOS := ...;                 (* local SubNet QoS *)

     HOST_A_MASK = 2**24-1;           (* mask for CLASS A host addr *)
     HOST_B_MASK = 2**16-1;           (* mask for CLASS B host addr *)
     HOST_C_MASK = 2**8-1;            (* mask for CLASS C host addr *)

     NET_A_MASK = 2**31-1;            (* mask for CLASS A net addr *)
     NET_B_MASK = 2**30-1;            (* mask for CLASS B net addr *)
     NET_C_MASK = 2**29-1;            (* mask for CLASS C net addr *)

     NET_A_SHIFT = 2**24;             (* r-shift for CLASS A net addr *)
     NET_B_SHIFT = 2**16;             (* r-shift for CLASS B net addr *)
     NET_C_SHIFT = 2**8;              (* r-shift for CLASS C net addr *)

     CLASS_A_FLAG = 2**31;            (* CLASS A host addr limit *)
     CLASS_B_FLAG = 2**31+2**30;      (* CLASS B host addr limit *)
     CLASS_C_FLAG = 2**31+2**30+2**29; (* CLASS C host addr limit *)

     D_IP_VERSION = 4;

     DONT_FRAGMENT = 64;
     MORE_FRAGMENT = 32;

     ICMP_PROTO = 1;                  (* protocol field *)

     OPTION_COPIED_MASK = 128;        (* copied flag in opt_code *)
     OPTION_CLASS_MASK = 96;          (* option class in opt_code *)
     OPTION_NUMBER_MASK = 31;         (* option number opt_code *)

     OPTION_CLASS_SHIFT = 32;

     CLASS_CONTROL = 0;               (* option class *)
     CLASS_MEASURE = 2;
```

```
OPT_END = 0;                         (* option number *)
NO_OP = 1;
SECURITY = 2;
L_SOURCE_ROUTING = 3;
TIMESTAMP = 4;
RECORD_ROUTING = 7;
STREAM_ID = 8;
SOURCE_ROUTING = 9;

ICMP_ECHO_REPLY = 0;                 (* ICMP type *)
ICMP_DESTIN_UNREACHABLE = 3;
ICMP_SOURCE_QUENCH = 4;
ICMP_REDIRECT = 5;
ICMP_ECHO = 8;
ICMP_TIME_EXCEEDED = 11;
ICMP_PARAM_PROBLEM = 12;
ICMP_TIMESTAMP = 13;
ICMP_TIMESTAMP_REPLY = 14;
ICMP_INFO_REQUEST = 15;
ICMP_INFO_REPLY = 16;

ICMP_NET_UNREACHABLE = 0;            (* ICMP code *)
ICMP_HOST_UNREACHABLE = 1;
ICMP_PROTO_UNREACHABLE = 2;
ICMP_PORT_UNREACHABLE = 3;
ICMP_FRAGMENT_FAILED = 4;
ICMP_SOURCE_ROUTE_FAILED = 5;

ICMP_LIFETIME_EXCEEDED = 0;
ICMP_REASSEMBLY_EXCEEDED = 1;

FLAG_type = (OK, EXPIRED, INCOMPLETE, COMPLETE);


type

ICMP_type = record
     typ: octet;
     code: octet;
     checksum: short_word;
     data: data_type;
     end;
```

```
var
   D_IP_pair: queue_type of IP_pair_type;
   mod_id: octet;
   mes_id: integer;
   cur_D_IP_pdu,err_pdu: D_IP_PDU_type;
   cur_IP_TP_mes: IP_TP_message_type;
   cur_IP_mes: IP_message_type;
   cur_IP_handler: IP_message_handler_type;
   opt_ptr: D_IP_option_type;
   s_host_addr,d_host_addr: word_type;
   s_net_addr,d_net_addr: word_type;
   s_ip_addr,d_ip_addr: word_type;
   host_mask,net_mask: word_type;
   net_shift: word_type;
   s_route,r_route: option_type;
   D_SN_addr: SNi_addr_type;
   ICMP_ptr: ^ICMP_type;

   pair_index: integer;
   err_flag,err_code: octet;
   transit_time: octet;

   pdu_error: integer;
   icmp_unreachable: integer;
   icmp_quench: integer;
   icmp_exceeded: integer;
   icmp_parameter: integer;
```

```
(***                                                 ***)
(***              Common Procedures                  ***)
(***                                                 ***)

function trans_d_ip_addr(addr: word_type) : word_type; primitive;

          (* translate DDN_IP address to GW_IP address *)


function trans_g_ip_addr(addr: word_type) : word_type; primitive;

          (* translate GW_IP address to DDN_IP address *)


function trans_d_lifetime(lifetime: octet) : octet; primitive;

          (* translate DDN_IP lifetime to GW_IP lifetime *)


function trans_g_lifetime(lifetime: octet) : octet; primitive;

          (* translate GW_IP lifetime to DDN_IP lifetime *)


function trans_d_option(var opt_length: integer;
                        option: data_type;
                        pdu: D_IP_PDU_type) : data_type; primitive;

          (* translate DDN_IP options to GW_IP options *)


function trans_g_option(var opt_length: integer;
                        option: data_type;
                        pdu: D_IP_PDU_type) : data_type; primitive;

          (* translate GW_IP options to DDN_IP options *)

  (* NOTICE the differences of the options:
     PRECEDENCE, DELAY, THROUGHPUT, and RELIABILITY of DDN-IP
     are in the IP header, while ISO-IP keeps them in options *)


function one_option_length(code: octet) : integer;  primitive;

          (* get the length of one DDN-IP option *)
```

```
function conv_routing(opt_ptr: D_IP_option_type) :
                                 routing_option_type; primitive;

          (* convert the routing direction
             between the  SOURCE_ROUTING and RECORDED_ROUTING *)


procedure copy_options(pdu: D_IP_PDU_type);  primitive;

          (* copy options according to COPIED flag in options
             for those PDUs with offset>0 *)


procedure send_ICMP (err_type, err_code: octet; pdu: D_IP_PDU_type);
                                                       primitive;

          (* send an ICMP for echo *)
```

```
function get_data_unit_id : short_word;

        (* get unique Data_Unit_Id *)

    begin
    with D_IP_pair[pair.index] do
         begin
         unitid := unitid+1;
         get_data_unit_id := unitid;
         end;
    end;


function get_option_length(pdu: D_IP_PDU_type) : integer;

        (* get the length of options *)

    begin
    get_option_length := (pdu.I_length - 5) * 4;
    end;


function opt_num(code: octet) : integer;

        (* get the option number from the option code *)

    begin
    opt_num := code and OPTION_NUMBER_MASK;
    end;



procedure one_segment(pdu: D_IP_PDU_type);

        (* output one IP_PDU *)

    var
         i,j,k,leng,length: integer;
         s: word_type;

    begin
    with pdu do
         begin
         length := (I_length and 15) * 4;
         I_length := (I_length and 15) + D_IP_VERSION*16;
         tot_length := length + d_length(data);

         leng := get_option_length(pdu);
         opt_ptr := ^option;
```

```
            s_route := d_null;     (* looking for SOURCE_ROUTING *)
            while ((s_route=d_null) and (leng>0)) then
              if (opt_num(opt_ptr.code)=SOURCE_ROUTING) then
                 s_route:=d_gets(opt_ptr,1,opt_ptr.length);
               else
                 begin
                 k := one_option_length(opt_ptr.code);
                 leng := leng - k;
                 opt_ptr := ^opt_ptr + k;
                 end;
            if (s_route=d_null) then
                 s_route :=
                   conv_routing(D_IP_pair.buf[pair_index].r_routing);
            k := s_route.value[1]          (* get pointer to field *)
            s := get_word(^s_route.value[k-2], 4);
                                           (* get the IP adress *)
            D_SN_addr := trans_SN_addr(s); (* translate to SN_addr *)
            opt_ptr.value[1] := opt_ptr.value[1] + 4;
                                           (* modify the pointer *)
            checksum := 0;
            set_checksum (pdu, length, 11);
            end;

        output NET_INi.D_IP_PDUreq (D_SN_addr,
                                    D_SN_LOCAL_ADDR,
                                    D_SN_QOS,
                                    pdu);
        end;



    function pair_match : integer;

            (* match the D_IP_pair with s/d_ip_addr,
               set and return the index *)

        var
            i,j,k,leng: integer;
            t: word_type;

        begin
        j := 0;
        for i := 1 to D_IP_pair.last do    (* look for the pair with
                                              same IP_address info *)
          with D_IP_pair.buf[i] do
            if ((s_IP_addr = s_ip_addr) and
                (d_IP_addr = d_ip_addr)) then
                  j := i;

         if ((j<>0) and (mod_id=0)) then              (* matched *)
            mod_id := D_IP_pair.buf[j].module/16;
         else if ((j<>0) and (mod_id<>0) and      (* matched *)
                 (D_IP_pair.buf[j].module/16=0)) then
```

```
        D_IP_pair.buf[j].module := mod_id*16 + i_module_id;
    else if (j=0) then    (* need fill in new IP_pair entry *)
        begin
        if (D_IP_pair.last<MAX_QUEUE) then
             begin
             D_IP_pair.last := D_IP_pair.last+1;
             j := D_IP_pair.last;
             end;
         else                                   (* no empty slot *)
             begin
             t := cur_time();
             for i := 1 to D_IP_pair.last do
               if (D_IP_pair.buf[i].timestamp<t) then
                  begin
                  j := i;
                  t := D_IP_pair.buf[i].timestamp;
                  end;

             pair_index := j;
             for k := 1 to segment_que.last do
                                  (* clean reassembly queue *)
               with segment_que.buf[k] do
                  send_ICMP(ICMP_SOURCE_QUENCH, 0, pdu);
                                  (* send ICMP to report pdu lost *)
             end;

        with D_IP_pair.buf[j] do (* fill in *)
             begin                  (* use empty/oldest slot *)
             module = mod_id * 16 + i_module_id;
             s_IP_addr := s_ip_addr;
             d_IP_addr := d_ip_addr;
             s_routing := d_null;
             d_routing := d_null;
             rcv_mes_que.last := 0;
             send_mes_que.last := 0;
             segment_que.last := 0;
             timestamp = cur_time();
             end; (* with: fill in *)
        end;  (* if j=0 *)

   pair_index := j;
   pair_match := j;
   end;
```

```
(***                                              ***)
(***            initialization                    ***)
(***                                              ***)


initialize

     pdu_error := 0;
     icmp_unreachable := 0;
     icmp_quench := 0;
     icmp_exceeded := 0;
     icmp_parameter := 0;

     D_IP_pair.last := 0;
end;




(***                                              ***)
(***          State Transition                    ***)
(***                                              ***)


trans
  when NET_INi.D_IP_PDUind               (* from DDN subnetwork *)

     var
          i,ii,j,k,leng,length,total: integer;
          t: word_type;
          pdu1: D_IP_PDU_type;
          flag: FLAG_type;

     begin
     transit_time := cur_time();
     cur_D_IP_pdu := packet;

     s_ip_addr := cur_D_IP_pdu.source_addr;
     d_ip_addr := cur_D_IP_pdu.destin_addr;
     mod_id := 0;
     j := pair_match();
     err_flag := NO_ERROR;
     s_route := d_null;
     r_route := d_null;
     with cur_D_IP_pdu do
       with D_IP_pair.buf[j] do
          begin
          length := (I_length and 15) * 4;
          if(chk_checksum(cur_D_IP_pdu,length)=false) then
               begin
               err_flag := ICMP_PARAM_PROBLEM;
```

```
        err_num := 11;
        end;
    else
        begin
        lifetime := lifetime - D_LIFETIME;
        leng := get_option_length(cur_D_IP_pdu);
        opt_ptr := ^option;
        while (leng>0) then
             if (opt_num(opt_ptr.code)=SOURCE_ROUTING) then
                  begin
                  s_route:=d_gets(opt_ptr,1,opt_ptr.length);
                  s_routing := s_route;
                  end;
              else if (opt_ptr.code=RECORD_ROUTING) then
                  begin
                  r_route := conv_routing(opt_ptr);
                  r_routing := r_route;
                  end;
              else
                  begin
                  k := one_option_length(opt_ptr.code);
                  leng := leng - k;
                  opt_ptr := ^opt_ptr + k;
                  end;
        if((err_flag=NO_ERROR) and (lifetime<=0)) then
             begin
             err_flag := ICMP_TIME_EXCEEDED;
             err_num := ICMP_LIFETIME_EXCEEDED;
             end;
         else if((err_flag=NO_ERROR) and (s_route=d_null)) then
             begin
             err_flag := ICMP_DESTIN_UNREACHABLE;
             err_num := ICMP_SOURCE_ROUTING_FAILED;
             end;
        end;

     if (err_flag<>NO_ERROR) then  (* send Error/Report *)
        begin
        send_ICMP(err_flag, err_code, cur_D_IP_pdu);
        pdu_error := pdu_error+1;      (* PDU dropped *)
        end;
      else
        begin                          (* handle segment  *)
        flag := OK;
        ii := 0;
        if ((flags and MORE_FRAGMENT=0) and
            (offset=0))then
             begin
             pdu1 := cur_D_IP_pdu;   (* no segmentation *)
             k := tot_length - (I_length and 15) * 4;
             flag := COMPLETE;
             end;
         else
             begin
             i := 1;
```

```
      while ((i<segment_que.last) and
             (id<segment_que.buf[i].pdu.id)) do
            i := i + 1;
      ii := i;
      k := 0;
      if (offset=0) then       (* of cur_D_IP_pdu *)
           begin
           pdu1 := cur_D_IP_pdu;
           k := tot_length - (I_length and 15) * 4;
           end;
      while ((flag=OK) and
             (i=<segment_que.last) and
             (id=segment_que.buf[i].pdu.id)) do
        begin
        with segment_que.buf[i] do
           begin
           if (pdu.lifetime<=cur_time()- time) then
                flag := EXPIRED;
            else if ( k < pdu.offset*8) then
                flag := INCOMPLETE;
            else
                begin
                if (k=0) then
                     pdu1 := pdu;
                 else
                     d_append(pdu1.data,pdu.data);
                length := (pdu.I_length and 15)*4;
                k := k + pdu.tot_length - length;
                end;
           end;
        if (k=8*offset) then   (* of cur_D_IP_pdu *)
           begin
           d_append(pdu1.data, data);
           length := (I_length and 15)*4;
           k := k + tot_length - length;
           end;
        if (flag=OK) then
           i := i+1;
         else                  (* stop *)
           i := segment_que.last+1;
        end;  (* while *)
      end;  (* else *)
 if (((flag=OK) and (i=segment_que.last)) or
                                   (* miss last seg.*)
     (flag=INCOMPLETE)) then
                       (* PDU is not complete yet *)
      begin            (* queue cur_D_IP_pdu *)
      for k := segment_que.last to ii do
           segment_que.buf[k+1] := segment_que.buf[k];
      segment_que.buf[ii].pdu := cur_D_IP_pdu;
      segment_que.buf[ii].time := cur_time();
      segment_que.last := segment_que.last+1;
      end;
  else if (ii>0) then  (* need to extract segments
                            expired/reassemblied *)
```

```
            while ((ii<segment_que.last) and
                  (id=segment_que.buf[ii].pdu.id)) do
                 begin
                 exqueue(segment_que, ii);
                 ii := ii+1;
                 end;

       if (flag=EXPIRED) then
            send_ICMP(ICMP_TIME_EXCEEDED,
                      ICMP_REASSEMBLY_EXCEEDED,
                      cur_D_IP_pdu);
        else if(flag=COMPLETE) then
            begin
            with cur_IP_TP_mes do
                 begin
                 module := D_IP_pair.buf[j].module;
                 mes_id := mes_id + 1;
                 if (mes_id=0) then
                      mes_id := 1;
                 message := mes_id;
                 s_IP_addr := s_ip_addr;
                 d_IP_addr := d_ip_addr;
                 data := pdu1.data;
                 end;
            output NET_OUTi.D_IP_PDUind (cur_IP_TP_mes);

            with cur_IP_mes do
                 begin           (* make GW packet *)
                 module := D_IP_pair.module;
                 message := mes_id;
                 source_addr := trans_d_ip_addr(s_ip_addr);
                 destin_addr := trans_d_ip_addr(d_ip_addr);
                 lifetime := trans_d_lifetime(pdu1.lifetime)
                             - (cur_time()-transit_time);
                 option := trans_d_option(pdu1.option);
                 end;
            output NET_GWi.IP_MESind (cur_IP_mes);

            with cur_IP_handler do
                 begin      (* make IP_message_handler *)
                 message := mes_id;
                 lifetime := pdu.lifetime;
                 option := pdu.option;
                 end;
            enqueue (rcv_mes_que, cur_IP_handler);

            end;  (* if flag *)
       end;  (* else err_flag *)

     timestamp := cur_time;
     end;  (* with-with *)
end;
```

```
trans
  when NET_GWi.IP_MESreq             (* from GW common linker *)

      var
           j,leng,length: integer;

      begin
      cur_IP_mes := packet;
      with cur_IP_mes do
           begin
           mod_id := module/16 + (module mod 16)*16;
           d_ip_addr := trans_g_ip_addr(source_addr);
           s_ip_addr := trans_g_ip_addr(destin_addr);
           j := pair_match();
           with D_IP_pair.buf[j] do
                begin
                s_route := s_routing;
                r_route := r_routing;
                if ((s_route=d_null) or (r_route=d_null)) then
                     begin
                     opt_ptr := ^option;
                     leng := d_length(option);
                     while (leng>0) then
                       if ((r_route=d_null) and
                           (opt_num(opt_ptr.code)=SOURCE_ROUTING)) then
                          begin
                          r_route:= conv_routing(opt_ptr);
                          r_routing := r_route;
                          end;
                      else if ((s_route=d_null) and
                               (opt_ptr.code=RECORD_ROUTING)) then
                          begin
                          s_route := d_gets(opt_ptr,1,opt_ptr.length);
                          s_routing := s_route;
                          end;
                      else
                          begin
                          k := one_option_length(opt_ptr.code);
                          leng := leng - k;
                          opt_ptr := ^opt_ptr + k;
                          end;
                 end; (* if *)
                with cur_IP_handler do
                     begin      (* make IP_message_handler *)
                     message := mes_id;
                     lifetime := trans_g_lifetime(cur_IP_mes.lifetime);
                     option := trans_g_option(cur_IP_mes.option);
                     in_time := cur_time();
                     end;
                enqueue (send_mes_que, cur_IP_handler);
                timestamp := cur_time;
                end;  (* with D_IP_pair.buf[j] *)
           end;  (* with cur_IP_mes *)
       end;
```

```
trans
  when NET_OUTi.D_IP_PDUreq                          (* from TCP *)

     var
          i,j,k,n: integer;
          handler: IP_message_handler_type;
          ip_data: data_type;

     begin
     cur_IP_TP_mes := packet;
     with cur_IP_TP_mes do
          begin
          mod_id := module/16 + (module mod 16)*16;
          mes_id := message;
          s_ip_addr := d_IP_addr;
          d_ip_addr := s_IP_addr;

          j := 0;
          for i := 1 to D_IP_pair.last do
            with D_IP_pair.buf[i] do
               if ((s_ip_addr=s_IP_addr) and
                   (d_ip_addr=d_IP_addr)) then
                        j := i;

          if (j=0) then
               pdu_error := pdu_error+1;
                    (* It is supposed to have a match here
                       because even if the IP_module at the
                       gateway side initiated the session, the
                       IP_message has been transferred over
                       faster to establish an entry earlier. *)
           else
             with D_IP_pair.buf[j] do
               begin
               pair_index := j;                (* global *)
               k := 0;
               if (mes_id<>0) then
                 for i := 1 to send_mes_que.last do
                   if (send_mes_que.buf[i].message=mes_id) then
                     k := i;
               if (k=0) then
                    k := 1;       (* if no match, use 1st one *)
               handler := exqueue(send_mes_que, k);
               with cur_D_IP_pdu do
                    begin
                    service := 0;
                    destin_addr := s_ip_addr;
                    source_addr := d_ip_addr;
                    id := get_data_unit_id;
                    flags := 0;
                    offset := 0;
                    protocol := TCP_PROTOCOL;
                    lifetime := handler.lifetime
                           - (cur_time() - handler.in_time);
```

```
                    option := handler.option;
                    I_length := d_length(option)/4 + 5;
                    data := cur_IP_TP_mes.data;

                                   (* segmentation & output *)
                    n := d_length(data);
                    if (n > D_IP_MAX_SEGMENT_LENGTH) then
                         begin
                         ip_data := data;
                         i := 1;
                         while (n>i*D_IP_MAX_SEGMENT_LENGTH) do
                              begin
                              flags := flags or MORE_FRAGMENT;
                              offset := offset +
                                        D_IP_MAX_SEGMENT_LENGTH/8;
                              data := d_gets(ip_data,
                                      (i-1)*D_IP_MAX_SEGMENT_LENGTH+1,
                                      D_IP_MAX_SEGMENT_LENGTH);
                              if(i=2) then
                                   copy_options();
                              one_segment(cur_D_IP_pdu);
                              i := i+1;
                              n := d_length(ip_data);
                              end;
                         flags := flags and (not MORE_FRAGMENT);
                         data:=d_gets(ip_data,
                               (i-1)*D_IP_MAX_SEGMENT_LENGTH+1,
                               d_length(ip_data)
                                      -D_IP_MAX_SEGMENT_LENGTH);
                         end;  (* if n > *)

                    one_segment();
                    end;  (* with cur_D_IP_pdu *)
               timestamp := cur_time;
               end;  (* with D_IP_pair.buf[j] *)
          end;  (* with cur_IP_TP_mes *)
     end;  (* trans *)

end;  (* of DDN_IP_entity_body *)
```

APPENDIX A.3 TRANSPORT_LAYER MODULE SPECIFICATION

```
(***                                                    ***)
(***      TRANSPORT_LAYER Module Specification          ***)
(***                                                    ***)

body TRANSPORT_LAYER_entity_body for TRANSPORT_LAYER_entity_type;

  constant

     MAX_CONN = ...;             (* Max Transport Connections *)

                                 (* position of varriable part *)
     CR_VAR = 7;                 (* for CR, CC, DR *)
     DC_VAR = 6;                 (* for DC *)
     DT_VAR = 5;                 (* for DT, ED, AK, EA, ER *)
     DTe_VAR = 8;                (* for DTe, EDe, AKe, EAe *)

     EDCODE = 1;                 (* TPDU code *)
     EACODE = 2;
     RJCODE = 5;
     AKCODE = 6;
     ERCODE = 7;
     DRCODE = 8;
     DCCODE = 12;
     CCCODE = 13;
     CRCODE = 14;
     DTCODE = 15;

     CLASS_0 = 0;                (* Class of Transport Service *)
     CLASS_1 = 1;
     CLASS_2 = 2;
     CLASS_3 = 3;
     CLASS_4 = 4;

     CALINGCODE = 193;           (* Calling TSAP ID *)
     CALLEDCODE = 194;           (* Called TSAP ID *)
     PDUSIZCODE = 192;           (* max TPDU size, 7..13 = 128..8,192 *)
     VERSNOCODE = 196;           (* version No., value=1 *)
     SECURICODE = 197;           (* user-def protection parameter *)
     CHKSUMCODE = 195;           (* checksum, in CLASS_4 only *)
     ADDLOPCODE = 198;           (* additional options,
                                      2:no checksum, 1:use ED *)
     ALTCLSCODE = 199;           (* alternative Class *)
     AKTIMECODE = 133;           (* short_word of max ack time, in ms *)
     THRUPTCODE = 137;           (* throughput *)
     RESERRCODE = 134;           (* residual error *)
     PRIORTCODE = 135;           (* priority *)
     TRNDELCODE = 136;           (* Transit delay *)
     REASSGCODE = 139;           (* reassignment time *)
```

```
SUBSEQCODE = 138;          (* sub-sequence number *)
FLOCONCODE = 140;          (* flow control confirmation *)


R_not_specified = 0;       (* reasons *)
R_TSAP_congestion = 1;
R_address_unknown = 3;
R_user_normal = 128;
R_peer_congestion = 129;
R_negotiation_failed = 130;
R_duplicate_src_ref = 131;
R_ref_mismatched = 132;
R_protocol_error = 133;
R_not_used = 134;
R_ref_overflow = 135;
R_CR_refused = 136;
R_not_used = 137;
R_invalid_hdr_param_length = 138;



type

  retrans_type = record
       timer : integer;     (* time left before retrans *)
       count : integer;     (* count of retrans *)
       data  : data_type;   (* PDU *)
       end;


  reorder_type = record
       seq  : integer;      (* sequence number for re-order *)
       data : data_type;    (* PDU *)
       end;


  TP_param_type = (CALINGCODE, CALLEDCODE, PDUSIZCODE, VERSNOCODE,
                   SECURICODE, CHKSUMCODE, ADDLOPCODE, ALTCLSCODE,
                   AKTIMECODE, THRUPTCODE, RESERRCODE, PRIORTCODE,
                   TRNDELCODE, REASSGCODE);


  TP_var _part_type = record
       param_code : TP_param_type; (* parameter code *)
       length     : octet;         (* length indicator *)
       value      : data_type;     (* parameter value *)
       end;
         (* the data structure defined here is for the
            variable part of the TPDU.  Some kinds of TPDUs
            can have multiple parameters whose total length
            can be decided indirectly by the length of the
            TPDU header. *)
```

```
throughput_type = record
                                        (* maximum *)
     max_AB : integer;                  (* max. calling - called *)
     min_AB : integer;                  (* min. *)
     max_BA : integer;                  (* max. called - calling *)
     min_BA : integer;                  (* min. *)
                                        (* average *)
     ave_AB : integer;
     a_min_AB: integer;
     ave_BA : integer;
     a_min_BA: integer;
     end;


residual_error_type = record
     target : integer;
     min_acc : integer;
     TSDU_size : integer;
     end;


transit_delay_type = record
     target_AB : integer;
     max_AB : integer;
     target_BA : integer;
     max_BA : integer;
     end;


TPDU_code_type = (EDCODE, EACODE, RJCODE, AKCODE, ERCODE,
                  DRCODE, DCCODE, CCCODE, CRCODE, DTCODE);


TP_class_type = (CLASS_0, CLASS_1, CLASS_2, CLASS_3,
                 CLASS_4);


TP_reason_type = (R_not_specified, R_TSAP_congestion,
                  R_address_unknown, R_user_normal,
                  R_peer_congestion, R_negotiation_failed,
                  R_duplicate_src_ref, R_ref_mismatched,
                  R_protocol_error, R_not_used,
                  R_ref_overflow, R_CR_refused,
                  R_not_used, R_invalid_hdr_param_length);
```

```
TPDU_message_type = record
     module      : octet;              (* Module Id *)
     message     : octet;              (* Message Id *)
     s_IP_addr   : word_type;          (* source IP address *)
     d_IP_addr   : word_type;          (* destin IP address *)
     tpdu_code   : TPDU_code_type;     (* TPDU code *)
     dst_ref     : short_word;         (* GW desti_refrence *)
     src_ref     : short_word;         (* GW source_refrence *)
     class       : TP_class_type;      (* Class option *)
     reason      : TP_reason_type;     (* reason/reject_cause *)
     v_length    : octet;              (* var_part length *)
     var_part    : TP_var_part_type;   (* variable part *)
     d_length    : integer;            (* data length *)
     data        : data_type;          (* user data *)
     end;
```

(* The data structure defined above is used for the communication between the Transport Layer protocol-dependent submodules and the common linker module.

One point to notice is that the format is close to the ISO TPDU structure. It is chosen because the ISO TP data structures have more capacity than those of DoD TCP, except for the Urgent Data. So it is quite reasonable to use the format closer to the standard one. The Urgent Data problem will be solved in some strategic way such as using the Expedited Data to transfer the Urgent Data.

Another point to notice is that both the sequence number and the credit are removed from the data structure. The reason behind is that these two control parameters are local significant only. Each side of the gateway will have independent control over them. *)

```
(***                                                        ***)
(***            Module Interaction Points                   ***)
(***                                                        ***)


(* TP Transport Gateway Interaction Point (to linker) *)


channel TP_linker_primitives (user, provider);

  by user:
     TP_MESreq (message : TPDU_message_type);

  by provider:
     TP_MESind (message : TPDU_message_type);



(***                                                        ***)
(***              Submodule Definitions                     ***)
(***                                                        ***)


module ISO_TP4_entity_type process
     (NSi:  NCEP_primitives (user,provider);
      TGi: TP_linker_primitives (user,provider);
     );
   export
     tp4_module_id: integer;    (* ISO TP module id *)
     inet_sent: integer;        (* packets sent to ISO IP *)
     inet_rcved: integer;       (* packets received from ISO IP *)
     inet_error: integer;       (* errors in packets with ISO IP *)
     end;


module DDN_TCP_entity_type process
     (NSd:  NCEP_primitives (user,provider);
      TGd: TP_linker_primitives (user,provider);
     );
   export
     tcp_module_id: integer;    (* DDN TCP module id *)
     dnet_sent: integer;        (* packets sent to DDN IP *)
     dnet_rcved: integer;       (* packets received from DDN IP *)
     dnet_error: integer;       (* errors in packets with DDN IP *)
     end;

body ISO_TP4_entity_body for ISO_TP4_entity_type; external;
body DDN_TCP_entity_body for DDN_TCP_entity_type; external;
```

```
(***                                                 ***)
(***            initialization                       ***)
(***                                                 ***)

initialize

     init ISO_TP4 with ISO_TP4_entity_body();
     attach NSi to ISO_TP4.NSi;                (* with ISO IP *)
     connect TGi to ISO_TP4.TGi;
     ISO_TP4.tp4_module_id := 9;
     ISO_TP4.inet_rcved := 0;
     ISO_TP4.inet_sent := 0;
     ISO_TP4.inet_error := 0;

     init DDN_TCP with DDN_TCP_entity_body();
     attach NSd to DDN_TCP.NSd;                (* with DDN IP *)
     connect TGd to DDN_TCP.TGd;
     DDN_TCP.tcp_module_id := 10;
     DDN_TCP.dnet_rcved := 0;
     DDN_TCP.dnet_error := 0;
     DDN_TCP.dnet_error := 0;
     end;



(***                                                 ***)
(***          State Transition                       ***)
(***                                                 ***)

trans
  when TGi.TP_MESind       (* from ISO_TP4 to Common Linker *)

     begin
     output TGd.TP_MESreq(packet);  (* send to DDN_TCP *)
     end;


trans
  when TGd.TP_MESind       (* from DDN_TCP to Common Linker *)

     begin
     output TGi.TP_MESreq(packet);  (* send to ISO_TP4 *)
     end;


end;      (* The End of TRANSPORT_LAYER module (Common Linker) *)
```

APPENDIX A.3.1 ISO_TP4 SUBMODULE

```
  (***                                                   ***)
  (***        ISO_TP4 Submodule Specification            ***)
  (***                                                   ***)

body ISO_TP4_entity_body for ISO_TP4_entity_type;




  (***                                                   ***)
  (***        TP4 Interface Interaction Point            ***)
  (***                                                   ***)

  (* TP-4 State Machine Interaction Point *)

channel TP4_machine_primitives (user, provider);

  by privider:
     TP4M_MESind (message : IP_TP_message_type);
                                  (* send TPDUs to TP4_machine *)

  by user:
     TP4M_MESreq (message : IP_TP_message_type);
                                  (* TP4_machine sends TPDUs out *)



  (* TP Transport Gateway Interaction Point (to linker module) *)

channel TP4_interface_primitives (user, provider);

  by provider:
     TP4I_MESind (message : IP_TP_message_type);

  by user:
     TP4I_MESreq (message : IP_TP_message_type);
```

```
(***                                                        ***)
(***    TP4 State Machine SubModule Specification           ***)
(***                                                        ***)


module TP4_machine_type process
         (TP4M_I: TP4_machine_primitives (user);
          TP4M_O: TP4_machine_primitives (provider);
          TP4I_I: TP4_interface_primitives (user);
          TP4I_O: TP4_interface_primitives (provider);
         );
     export
         state      : TP4_state_type; (* connection state *)
         l_module   : octet;       (* local module id *)
         g_module   : octet;       (* GW module id *)
         d_ip_addr  : word_type;   (* destination IP address *)
         s_ip_addr  : word_type;   (* source IP address *)
         l_ref      : short_word;  (* local reference *)
         r_ref      : short_word;  (* remote reference *)
         g_ref      : short_word;  (* gateway cross reference *)
         ex_flag    : boolean;     (* extended format *)
         ch_flag    : boolean;     (* checksum flag *)
         ed_flag    : boolean;     (* expedited data service *)
         class      : octet;       (* class and flags *)
         end;


body TP4_machine_body for TP4_machine_type;  external;
```

```
(***                                             ***)
(***    variables of the ISO_TP4_body             ***)
(***                                             ***)

var

    total_conn : integer;     (* total active transport conn *)
    conn_flag  : array [1..MAX_CONN] of boolean;
                              (* slot available flag *)
    TP4M       : array [0..MAX_CONN] of TP4_machine_type;
                              (* instance of TP4_machine *)

    cur_conn: integer;        (* cur connection reference *)
    direction : direction_type; (* cur mes flow direction *)
    cur_mod_id : octet;       (* module Id *)
    cur_mes_id : octet;       (* message Id *)
    cur_mes : IP_TP_message_type; (* cur IP_TP message *)
    cur_tpdu : TPDU_type;     (* TPDU, used as pointer *)
    cur_s_IP : word_type;     (* source IP address *)
    cur_d_IP : word_type;     (* destin IP address *)
    cur_tpdu_code : TPDU_code_type; (* current TPDU code *)
    flag_extended: boolean;   (* flag for extended format *)
    calling_tsap,called_tsap: short_word;

    total_mes : integer;      (* total messages going through *)
    err_mes : integer;        (* messages with error  conditions *)
    fatal_err : integer;      (* fatal errors *)
```

```
(***                                                     ***)
(***  Common Procedures of ISO_TP4_body               ***)
(***                                                     ***)


function conn_alloc : integer;

     (* allocate a free slot of connection *)

  var
     i,j : integer;

     begin
     j := 0;
     if (total_conn < MAX_CONN) then
       for i := MAX_CONN to 1 do
         if (conn_flag[i] = false) then
           j := i;
     if (j > 0) then
          begin
          total_conn := total_conn+1;
          conn_flag[j] := true;
          end;
     conn_alloc := j;
     end;
```

(* In the case of (total_conn=MAX_CONN), (conn_alloc=0) will be returned. There exists a TP4M[0] to be used in this case to respond to abnormal TPDUs. *)

```
procedure conn_free (j : integer);  (* free the connection slot *)

     begin
     if ((j>0) and (j<=MAX_CONN)) then
       if (conn_flag[j] = true) then
          begin
          conn_flag[j] := flase;
          total_conn := total_conn-1;
          end;
     end;


function TP4_code (TPDU: TPDU_type): TPDU_code_type;

     (* check the TPDU_code of the TPDU *)

     begin
     TP4_code := TPDU[2] / 16;                  (* 4 bits at left *)
     end;
```

```
function cur_TP4_code (mes: IP_TP_message) : TPDU_code_type;

     (* check the TPDU_code of the 1st IP_TP message in queue *)

     begin
     cur_mes := mes;                                (* access NQ message *)
     cur_tpdu := cur_mes.data;                      (* point to TPDU *)
     cur_conn := cur_tpdu.dst_ref;                  (* TCP reference index *)
     cur_tpdu_code := TP4_code(cur_tpdu);           (* get the TPDU code *)
     if (cur_tpdu_code<>CRCODE) then
          flag_extended := TP4M[cur_conn].ex_flag; (* ext.ed format *)
     cur_TP4_code := cur_tpdu_code;
     end;



function tpdu_fixed : integer;

     (* get the pointer to the variable part in cur_tpdu *)

  var
     i,l: integer;

     begin
     l := cur_tpdu[1];               (* get Length Indicator *)
     case cur_tpdu_code of
          CRCODE,CCCODE,DRCODE:
               i := CR_VAR;          (* point to variable part *)
          DCCODE:
               i := DC_VAR;
          DTCODE,AKCODE,EDCODE,EACODE,ERCODE:
               begin
               i := DT_VAR;
               if (flag_extended) then
                  i := i+3;
               end;
     end;
     tpdu_fixed := i;
     end;
```

```
function tpdu_acceptable (mes: IP_TP_message_type) : boolean;

     (* check the TPDU to see if it is acceptable *)

  var
     ok,checksum: boolean;
     i,j,k,l: integer;

     begin
     checksum := false;
     ok := true;
     cur_TP4_code(mes);  (* get cur_conn, cur_TP_code, etc. *)

     if (cur_conn<>0) then
        begin
        if (con_flag[cur_conn]=false) then
          ok := false;
         else
          begin
          if (TP4M[cur_conn].g_module=0) then
               TP4M[cur_conn].g_module = cur_mes.module/16;
            else
               if(TP4M[cur_conn].g_module<>cur_mes.module/16) then
                    ok := flase;

          if (TP4M[cur_conn].d_ip_addr=0) then
               TP4M[cur_conn].d_ip_addr:=cur_mes.d_IP_addr;
            else
               if(TP4M[cur_conn].d_ip_addr<>cur_mes.d_IP_addr)
                 then
                    ok := flase;

          if (TP4M[cur_conn].s_ip_addr=0) then
               TP4M[cur_conn].s_ip_addr:=cur_mes.s_IP_addr;
            else
               if(TP4M[cur_conn].s_ip_addr<>cur_mes.s_IP_addr)
                 then
                    ok := flase;

          if(TP4M[cur_conn].r_ref=0) then
               TP4M[cur_conn].r_ref := cur_tpdu.src_ref;
            else if (cur_tpdu.src_ref<>TP4M[cur_conn].r_ref) then
                    ok := false;             (* check the SRC_REF *)
          end;
        end;

     if ((ok) and (cur_tpdu_code <> RJCODE)) then
          begin
          i := tpdu_fixed();              (* length of fixed part *)
          l := cur_tpdu[1]-i+1;           (* No of bytes left,
                                             LI exclude itself *)
          i := i+1;                       (* point to variable part *)
          while (l>0) do
               begin                      (* check valid var. part *)
```

```
                if(cur_tpdu[i]=CHKSUMCODE) then
                     checksum := true;   (* there exists checksum *)
                j := cur_tpdu[i+1];      (* get the length of field *)
                l := l-j-2;
                i := i+j+2;
                end;
           if ((ok) and (l<0)) then      (* error in variable part *)
                ok := false;

           if ((ok) and (checksum)) then
                ok := chk_checksum(cur_tpdu, cur_tpdu.length);
                                         (* check the checksum *)
           end;

      tpdu_acceptable := ok;
      end;



 function dup_CR (mes: IP_TP_message_type) : boolean;

      (* check the CR TPDU to see if it is duplicate *)

   var
      dup: boolean;
      CR_tpdu: TPDU_CR_type;
      i: integer;

      begin
      CR_tpdu := mes.data;
      cur_conn := 0;
      dup := false;
      i := 1;
      while ((i<=MAX_CONN) and (dup=false)) do
           begin
           if ((conn_flag[i]=true) and
               (mes.s_IP_addr=TP4M[i].s_ip_addr) and
               (mes.d_IP_addr=TP4M[i].d_ip_addr) and
               (CR_tpdu.src_ref=TP4M[i].r_ref)) then
                     begin
                     cur_conn := i;
                     dup := true;
                     end;
           i := i+1;
           end;
      dup_CR := dup;
      end;
```

```
function dup_CRg (mes: TPDU_message_type) : boolean;

    (* check the CR TPDU from TP_GW to see if it is duplicate *)

  var
    dup: boolean;
    i: integer;

    begin
    cur_conn := 0;
    dup := false;
    i := 1;
    while((i<=MAX_CONN) and (dup=false)) do
         begin
         if ((conn_flag[i]=true) and
             (mes.d_IP_addr=TP4M[i].s_ip_addr) and
             (mes.s_IP_addr=TP4M[i].d_ip_addr) and
             (mes.src_ref=TP4M[i].g_ref)) then
                  begin
                  cur_conn := i;
                  dup := true;
                  end;
         i := i+1;
         end;
    dup_CRg := dup;
    end;
```

```
(***                                           ***)
(***     Initialization of ISO_TP4_body        ***)
(***                                           ***)

initialize

     total_mes := 0;            (* total messages *)
     err_mes := 0;              (* error messages *)
     fatal_err := 0;            (* fatal errors *)

     total_conn := 0;           (* total transport connections *)
     for i := 1 to MAX_CONN do
          conn_flag := false; (* the connection slot is free *)

     init TP4M[0] with TP4_machine_interface_body();
                                (* virtual State_machine for
                                   responding to error CR's *)
     TP4M[0].l_ref := 0;
     connect TP4M_I to TP4M[0].TP4M_I;
     connect TP4I_I to TP4M[0].TP4I_I;
     attach TP4M_O to TP4M[0].TP4M_O;
     attach TP4I_O to TP4M[0].TP4I_O;

     end;
```

```
(***                                                  ***)
(***           State Transitions                      ***)
(***                                                  ***)

trans
  when NSi.IP_TP_MESind priority 0        (* receive from ISO-IP *)

     begin
     total_mes := total_mes+1;
     direction := in;

     if (tpdu_acceptable(message)=false) then
          err_mes := err_mes+1;           (* TPDU not acceptable *)
       else if (cur_tpdu_code=CRCODE) then  (* and cur_conn=0 *)
          begin                           (* Connection Request *)
          if (dup_CR(message)=false) then
               begin
               cur_conn := conn_alloc();
               if (cur_conn>0) then
                    begin                 (* initialize TP4 protocol
                                             machine *)
                    init TP4M[cur_conn] with
                                   TP4_machine_interface_body();
                    with TP4M[cur_conn] do
                         begin
                         l_ref := cur_conn;
                         flag_kill_me := conn_in_progress;
                         end;
                    connect TP4M_I to TP4M[cur_conn].TP4M_I;
                    connect TP4I_I to TP4M[cur_conn].TP4I_I;
                    attach TP4M_O to TP4M[cur_conn].TP4M_O;
                    attach TP4I_O to TP4M[cur_conn].TP4I_O;
                    end;  (* if conn *)
               end; (* if dup_CR *)
          output TP4M[cur_conn].TP4M_I(message);
                (* pass TPDU for further check or response *)
          end;  (* if CR *)
       else
          output TP4M[cur_conn].TP4M_I(message);
     end;  (* trans *)

  (* For duplicated CR, the state machine TP4M[cur_conn] will
     response to it with CC, as required by ISO 8073.

     In case there is no local connection slot available,
     cur_conn is reset to 0 by dup_CR(), or by conn_alloc(), the
     CR will be passed to TP4M[0] which will in turn send a DR to
     refuse the connection request. *)
```

```
trans
  when TGS.TP_MESreq priority 0     (* receive from TCP, begore
                                       feeding into  TP4_machine *)

     begin
     total_mes := total_mes+1;
     direction := out;
     cur_mes := message;
     cur_tpdu := cur_mes.data;                   (* point to TPDU *)
     cur_conn := cur_tpdu.src_ref;               (* TCP reference index *)
     flag_extended := TP4M[cur_conn].class and 2; (* extended format *)
     cur_tpdu_code := TP4_code(cur_tpdu);        (* get the TPDU code *)
     case cur_tpdu_code of
       CCCODE,DRCODE,DCCODE,DTCODE,
       AKCODE,EDCODE,EACODE,ERCODE,
       RJCODE :
          output TP4M[cur_conn].TP4I_I(message);

       CRCODE :
          begin
          if (dup_CRg(message)=false) then
               begin
               cur_conn := conn_alloc();
               if (cur_conn>0) then
                    begin     (* initialize a new TP4_machine *)
                    init TP4M[cur_conn] with
                                 TP4_machine_interface_body();
                    with TP4M[cur_conn] do
                         begin
                         l_ref := cur_conn;
                         flag_kill_me := conn_in_progress;
                         end;
                    connect TP4M_I to TP4M[cur_conn].TP4M_I;
                    connect TP4I_I to TP4M[cur_conn].TP4I_I;
                    attach TP4M_O to TP4M[cur_conn].TP4M_O;
                    attach TP4I_O to TP4M[cur_conn].TP4I_O;
                    end;  (* if conn *)
               end; (* if *)
          output TP4M[cur_conn].TP4I_I(message);
                                (* pass TPDU for further checking *)
          end;  (* case CR *)
       end;  (* case *)
     end;  (* trans *)
```

```
trans
  any TP4M[i]: TP4_machine_interface_type do
    provided (TP4M[i].flag_kill_me = now) priority 1

     begin
     disconnect TP4M[i].TP4M_I;  (* disconnect port relations *)
     disconnect TP4M[i].TP4I_I;
     disattach TP4M[i].TP4M_O;
     disattach TP4M[i].TP4I_O;
     release TP4M[i];            (* terminate TP4_machine *)
     conn_free(i);               (* free the slot for future use *)
     end;

end;  (* of ISO_TP4_body *)
```

```
(***                                                     ***)
(***    TP4_machine_interface_body  Specification        ***)
(***                                                     ***)


body TP4_machine_interface_body for TP4_machine_interface_type;


constant

    VERSION = 1;                     (* TP-4 version, IS 8073 *)


type

  (* TPDU formats *)

    TPDU_type: data_type;                 (* general form *)


    TPDU_CR_type = record                 (* for CR, CC *)
         length     : octet;              (* length indicator *)
         tpdu_code  : TPDU_code_type;     (* TPDU code and CDT *)
         dst_ref    : short_word;         (* destin_refrence *)
         src_ref    : short_word;         (* source_refrence *)
         class      : TP_class_type;      (* Class option *)
         var_part   : TP_var_part_type;   (* variable part *)
         data       : data_type;          (* user data *)
         end;


    TPDU_DR_type = record                 (* for DR *)
         length     : octet;              (* length indicator *)
         tpdu_code  : TPDU_code_type;     (* TPDU code *)
         dst_ref    : short_word;         (* destin_refrence *)
         src_ref    : short_word;         (* source_refrence *)
         reason     : TP_reason_type;     (* reason *)
         var_part   : TP_var_part_type;   (* variable part *)
         data       : data_type;          (* user data *)
         end;


    TPDU_DC_type = record                 (* for DC *)
         length     : octet;              (* length indicator *)
         tpdu_code  : TPDU_code_type;     (* TPDU code *)
         dst_ref    : short_word;         (* destin_refrence *)
         src_ref    : short_word;         (* source_refrence *)
         var_part   : TP_var_part_type;   (* variable part *)
         end;
```

```
TPDU_DT_type = record                     (* for DT, ED *)
     length     : octet;                  (* length indicator *)
     tpdu_code  : TPDU_code_type;         (* TPDU code *)
     dst_ref    : short_word;             (* destin_refrence *)
     seq_no     : octet;                  (* sequence no and EOT *)
     var_part   : TP_var_part_type;       (* variable part *)
     data       : data_type;              (* user data *)
     end;

TPDU_DTe_type = record                    (* for extended DT, ED *)
     length     : octet;                  (* length indicator *)
     tpdu_code  : TPDU_code_type;         (* TPDU code *)
     dst_ref    : short_word;             (* destin_refrence *)
     seq_no     : word_type;              (* sequence no and EOT *)
     var_part   : TP_var_part_type;       (* variable part *)
     data       : data_type;              (* user data *)
     end;

TPDU_AK_type = record                     (* for AK, EA *)
     length     : octet;                  (* length indicator *)
     tpdu_code  : TPDU_code_type;         (* TPDU code and CDT *)
     dst_ref    : short_word;             (* destin_refrence *)
     seq_no     : octet;                  (* sequence no and EOT *)
     var_part   : TP_var_part_type;       (* variable part *)
     end;

TPDU_AKe_type = record                    (* for extended AK *)
     length     : octet;                  (* length indicator *)
     tpdu_code  : TPDU_code_type;         (* TPDU code *)
     dst_ref    : short_word;             (* destin_refrence *)
     seq_no     : word_type;              (* sequence no *)
     cdt        : short_word;             (* credit *)
     var_part   : TP_var_part_type;       (* variable part *)
     end;

TPDU_EAe_type = record                    (* for extended EA *)
     length     : octet;                  (* length indicator *)
     tpdu_code  : TPDU_code_type;         (* TPDU code *)
     dst_ref    : short_word;             (* destin_refrence *)
     seq_no     : word_type;              (* sequence no *)
     var_part   : TP_var_part_type;       (* variable part *)
     end;

TPDU_RJ_type = record                     (* for RJ *)
     length     : octet;                  (* length indicator *)
     tpdu_code  : TPDU_code_type;         (* TPDU code and CDT *)
     dst_ref    : short_word;             (* destin_refrence *)
     seq_no     : octet;                  (* sequence no and EOT *)
     end;
```

```
TPDU_RJe_type = record                  (* for extended RJ *)
     length     : octet;                (* length indicator *)
     tpdu_code  : TPDU_code_type;       (* TPDU code *)
     dst_ref    : short_word;           (* destin_refrence *)
     seq_no     : word_type;            (* sequence no *)
     cdt        : short_word;           (* credit *)
     end;


TPDU_ER_type = record                   (* for ER *)
     length     : octet;                (* length indicator *)
     tpdu_code  : TPDU_code_type;       (* TPDU code and CDT *)
     dst_ref    : short_word;           (* destin_refrence *)
     cause      : TP_cause_type;        (* Reject cause *)
     var_part   : TP_var_part_type;     (* variable part *)
     end;


mes_CR_type = record                    (* for CR, CC *)
     module    : octet;                 (* Module Id *)
     message   : octet;                 (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_CR_type;          (* CR TPDU *)
     end;


mes_DR_type = record                    (* for DR *)
     module    : octet;                 (* Module Id *)
     message   : octet;                 (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_DR_type;          (* DR TPDU *)
     end;


mes_DC_type = record                    (* for DC *)
     module    : octet;                 (* Module Id *)
     message   : octet;                 (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_DC_type;          (* DC TPDU *)
     end;


mes_DT_type = record                    (* for DT, ED *)
     module    : octet;                 (* Module Id *)
     message   : octet;                 (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_DT_type;          (* DT TPDU *)
     end;
```

```
mes_DTe_type = record                   (* for extended DT, ED *)
     module  : octet;                   (* Module Id *)
     message : octet;                   (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_DTe_type;         (* DTe TPDU *)
     end;

mes_AK_type = record                    (* for AK, EA *)
     module  : octet;                   (* Module Id *)
     message : octet;                   (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_AK_type;          (* AK TPDU *)
     end;

mes_AKe_type = record                   (* for extended AK *)
     module  : octet;                   (* Module Id *)
     message : octet;                   (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_AKe_type;         (* AKe TPDU *)
     end;

mes_EAe_type = record                   (* for extended EA *)
     module  : octet;                   (* Module Id *)
     message : octet;                   (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_EAe_type;         (* EAe TPDU *)
     end;

mes_RJ_type = record                    (* for RJ *)
     module  : octet;                   (* Module Id *)
     message : octet;                   (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_RJ_type;          (* RJ TPDU *)
     end;

mes_RJe_type = record                   (* for extended RJ *)
     module  : octet;                   (* Module Id *)
     message : octet;                   (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_RJe_type;         (* RJe TPDU *)
     end;
```

```
mes_ER_type = record                    (* for ER *)
     module  : octet;                   (* Module Id *)
     message : octet;                   (* Message Id *)
     s_IP_addr : word_type;             (* source IP address *)
     d_IP_addr : word_type;             (* destin IP address *)
     tpdu      : TPDU_ER_type;          (* ER TPDU *)
     end;


TP4_state_type = (CLOSED, OPEN, WFCCiso, WFCCout, AKWAITiso,
                  AKWAITout, CLOSINGiso, CLOSINGout, REFWAIT);
```

```
(***                                                  ***)
(***  Variables of TP4_machine_interface _body        ***)
(***                                                  ***)

var
    proto_err : integer;      (* fatal errors *)
    l_TSAP    : short_word;   (* local TSAP id *)
    r_TSAP    : short_word;   (* remote TSAP id *)
    initiator : initiator_type; (* who initiate the conn *)
    size      : octet;        (* max TPDU size *)
    vers      : octet;        (* TP version *)
    prot      : data_type;    (* user-def protection *)
    ack_time  : short_word;   (* max ack time in msec *)
    throuput  : throughput_type; (* throuput *)
    res_error : res_error_type; (* residual error *)
    priority  : short_word;   (* priority *)
    trans_delay: trans_delay_type; (* transit delay *)
    g_class   : octet;        (* GW class and flags *)
    g_size    : octet;        (* GW max TPDU size *)
    g_vers    : octet;        (* GW TP version *)
    g_prot    : data_type;    (* GW user-def protection *)
    g_ack_time: short_word;   (* GW max ack time in msec *)
    g_throuput: throughput_type; (* GW throuput *)
    g_res_error: res_error_type; (* GW residual error *)
    g_priority: short_word;   (* GW priority *)
    g_trans_delay: trans_delay_type; (* GW transit delay *)
    send_seq  : word_type;    (* sending sequence no *)
    send_cdt  : short_word;   (* sending credit *)
    s_sub_seq : short_word;   (* sub-seq no for AK *)
    s_ack_seq : word_type;    (* ack_ed sequence no *)
    rcv_seq   : word_type;    (* receiving sequence no *)
    rcv_cdt   : short_word;   (* receiving credit *)
    r_sub_seq : short_word;   (* sub-seq no for AK *)
    r_ack_seq : word_type;    (* ack_ed sequence no *)

    exp_flag  : boolean;      (* flag for ED blocking *)
    exp_seq   : integer;      (* ED sequence number *)
    exp_que   : queue_type of data_type; (* blocked DTs *)

    reason    : TP_reason_type; (* reason *)
    cause     : TP_cause_type;  (* Reject cause *)
    re_trans  : queue_type of retrans_type; (* retransmission *)
    re_order  : queue_type of reorder_type; (* re-order *)

    cur_mes : IP_TP_message_type; (* cur IP_TP message *)
    cur_tpdu : TPDU_type;      (* TPDU, used as pointer *)
    cur_tpdu_code : TPDU_code_type; (* current TPDU code *)
    var_ptr: ^TP_var_part_type;    (* ptr to variable part *)
    sub_state     : boolean;       (* associated flag *)
    seq : word_type;
    cdt : short_word;
```

```
mes CR,mes CC : mes CR type;  (* various kinds of message *)
mes DT,mes ED : mes DT type;
mes AK,mes EA : mes AK type;
mes DTe,mes EDe : mes DTe type;
mes AKe,mes EAe : mes AKe type;
mes DR : mes DR type;
mes DC : mes DC type;
mes RJ : mes RJ type;
mes RJe : mes RJe type;
mes ER : mes ER type;
ptr CR : TPDU CR type;          (* various kinds of ptrs *)
ptr DT : TPDU DT type;
ptr AK : TPDU AK type;
ptr DTe : TPDU DTe type;
ptr AKe : TPDU AKe type;
ptr DR : TPDU DR type;
ptr DC : TPDU DC type;
ptr RJ : TPDU RJ type;
ptr RJe : TPDU RJe type;
ptr ER : TPDU ER type;

tp mes : TPDU message type;
```

```
(***                                              ***)
(***          Common Procedures                   ***)
(***                                              ***)


procedure set_timer (FSM: TP4_machine_type;
                     time: integer;
                     kind: timer_type);     primitive;

     (* set timer *)


procedure set_retrans_timer;

     (* set re-trans timer *)

     begin
     set_timer (TP4M[l_ref], RETRANS_TIME, retrans);
     end;


procedure set_ref_timer;

     (* set ref timer *)

     begin
     set_timer (TP4M[l_ref], REF_TIME, ref);
     end;


procedure set_window_timer;

     (* set window timer *)

     begin
     set_timer (TP4M[l_ref], WINDOW_TIME, window);
     end;


procedure set_inact_timer;

     (* set inact timer *)

     begin
     set_timer (TP4M[l_ref], INACT_TIME, inact);
     end;
```

```
procedure stop_timer (FSM: TP4_machine_type;
                      kind: timer_type);     primitive;

     (* stop timer *)


procedure stop_retrans_timer;

     (* stop retrans timer *)

     begin
     stop_timer (TP4M[l_ref], retrans);
     end;


procedure stop_window_timer;

     (* stop window timer *)

     begin
     stop_timer (TP4M[l_ref], window);
     end;


procedure stop_inact_timer;

     (* stop inact timer *)

     begin
     stop_timer (TP4M[l_ref], inact);
     end;


function timer (kind: timer_type): logical;      primitive;

     (* called at the timer interrupt *)


function set_credit: integer;  primitive;

     (* set credit for local TP4M for receiving.  It is used to
         get the window width for the flow control.  The
         function can be implemented by estimating the available
         storage space in memory, dividing them between the
         modules in the system, and between the active sessions
         in the module. *)
```

```
function acceptable_param: logical;        primitive;

     (* check to see if the connection parameters
        are acceptable *)


function TP4_code (mes: IP_TP_message_type) : TPDU_code_type;

     (* check the TPDU_code of message from IP *)

     begin
     cur_mes := mes;
     cur_tpdu := cur_mes.data;
     cur_tpdu_code := cur_tpdu[2] / 16;       (* 4 bits at left *)
     TP4_code := cur_tpdu_code;
     end;


function TP_code (mes: TPDU_message_type) : TPDU_code_type;

     (* check the TPDU_code of message from TP-GW *)

     begin
     tp_mes := mes;
     cur_tpdu_code := tp_mes.tpdu_code;     (* 4 bits at right *)
     TP_code := cur_tpdu_code;
     end;


function tpdu_fixed : integer;

     (* get the pointer to the variable part in cur_tpdu *)

  var
     i,l: integer;

     begin
     l := cur_tpdu[1];                (* get Length Indicator *)
     case cur_tpdu_code of
          CRCODE,CCCODE,DRCODE:
               i := CR_VAR;           (* point to variable part *)
          DCCODE:
               i := DC_VAR;
          DTCODE,AKCODE,EDCODE,EACODE,ERCODE:
               begin
               i := DT_VAR;
               if (ex_flag) then
                  i := i+3;
               end;
       end;  (* case *)
     tpdu_fixed := i;
     end;
```

```
procedure tpdu_checksum (var mes: IP_TP_message_type);

    (* complete the TPDU with checksum, etc. *)

  var
    i,j,k,l: integer;

    begin
    i := tpdu_fixed();               (* length of fixed part *)
    l := cur_tpdu[1]-i+1;            (* No. of bytes left, LI
                                        exclude itself *)
    j := 0;
    i := i+1;                        (* point to variable part *)
    while ((l>0) and (j=0)) do
         begin
         if (cur_tpdu[i]=CHKSUMCODE) then
              j := i+2;              (* get pointer to checksum *)
           else
              begin
              k := cur_tpdu[i+1]; (* get the length of field *)
              l := l-k-2;
              i := i+k+2;
              end;
         end;
    if (j>0) then
         mes.data := set_checksum(cur_tpdu, cur_tpdu.length, j);
    end;
```

```
procedure accept_TPDU (mes: IP_TP_message_type);

    (* accept the control info in the TPDU from ISO-TP4 *)

    begin
    cur_mes := mes;
    cur_tpdu := cur_mes.data;
    cur_tpdu_code := cur_tpdu[2] / 16;      (* 4 bits at left *)

    sub_state := true;                    (* true for OK *)
    if (r_ref=0) then
         r_ref := cur_tpdu.src_ref;       (* fill in remote ref *)
      else if (r_ref<>cur_tpdu.src_ref) then
         sub_state := false;

    if (s_ip_addr=0) then
         s_ip_addr  := cur_tpdu.s_IP_addr;  (* fill in source IP *)
      else if (s_ip_addr<>cur_tpdu.s_IP_addr) then
         sub_state := false;

    if (d_ip_addr=0) then
         d_ip_addr  := cur_tpdu.d_IP_addr;  (* fill in destin IP *)
      else if (d_ip_addr<>cur_tpdu.d_IP_addr) then
         sub_state := false;

    case cur_tpdu_code of                 (* pick up special fields *)
     CRCODE,CCCODE :
         begin
         ptr_CR := ^cur_tpdu;
         class  := ptr_CR.class / 16;   (* left 4 bit for class *)
         if (ptr_CR.class and 2) then
              ex_flag := true;
           else
              ex_flag := false;
         if (cur_tpdu_code=CRCODE) then
              g_module := cur_mes.module/16;
         end;
      DRCODE :
         begin
         ptr_DR := ^cur_tpdu;
         reason := ptr_DR.reason;
         end;
      DTCODE,EDCODE :
         if (ex_flag=flase) then            (* normal *)
              begin
              ptr_DT := ^cur_tpdu;
              seq := get_word (ptr_DT.seq_no,1);
              end;
          else                              (* extended *)
              begin
              ptr_DTe := ^cur_tpdu;
              seq := ptr_DTe.seq_no;
              end;
      AKCODE,EACODE :
```

```
        if (ex_flag=flase) then              (* normal *)
             begin
             ptr_AK := ^cur_tpdu;
             seq := get_word (ptr_AK.seq_no,1);
             end;
         else                                (* extended *)
             begin
             ptr_AKe := ^cur_tpdu;
             seq := ptr_AKe.seq_no;
             end;
    RJCODE :
        if (ex_flag=flase) then              (* normal *)
             begin
             ptr_RJ := ^cur_tpdu;
             seq := get_word (ptr_RJ.seq_no, 1);
             end;
         else                                (* extended *)
             begin
             ptr_RJe := ^cur_tpdu;
             seq := ptr_RJe.seq_no;
             end;
    ERCODE :
       begin
       ptr_ER := ^cur_tpdu;
       cause := ptr_ER.cause;
       end;
    default :                                (* DC *)
       null;
  end;

  case cur_tpdu_code of                      (* pick up credit *)
    CRCODE,CCCODE :
       cdt := ptr_CR.tpdu_code mod 16;
    AKCODE :
       if (ex_flag=flase) then               (* normal *)
            cdt := ptr_AK.tpdu_code mod 16;
        else                                 (* extended *)
            cdt := ptr_AKe.cdt;
    RJCODE :
       if (ex_flag=flase) then               (* normal *)
            cdt := ptr_RJ.tpdu_code mod 16;
        else                                 (* extended *)
            cdt := ptr_RJe.cdt;
    default :
       null;
    end; (* case *)
  end;
```

```
procedure accept_TP4_param;

     (* accept the control parameters in the TPDU from ISO-TP4 *)

  var
     n: word_type;
     sn: short_word;
     i,j,k,l: integer;
     var_ptr: ^TP_var_part_type;

     begin
     i := tpdu_fixed();                   (* ptr to variable parts *)
     var_ptr := ^cur_tpdu[i+1];
     l := cur_tpdu[1]-i+1;                (* length of variable part *)
     while (l>0) do
          begin
          k := var_ptr.length;

          case var_ptr.param_code of
            CALINGCODE :         (* 1100 0001: calling TSAP-ID *)
               if (k>2) then
                    sub_state := false;
                 else if (cur_tpdu_code=CRCODE) then
                    r_TSAP := get_sword (var_ptr.value, k);
                 else if (cur_tpdu_code<>CCCODE) then
                    sub_state := false;
                 else if (r_TSAP=0) then
                    r_TSAP:=get_sword(var_ptr.value,k);
                 else if (r_TSAP<>get_sword(var_ptr.value,k))) then
                    sub_state := false;

            CALLEDCODE :         (* 1100 0010: called TSAP-ID *)
               if (k>2) then
                    sub_state := false;
                 else if (cur_tpdu_code=CRCODE) then
                    l_TSAP := get_sword (var_ptr.value, k);
                 else if (cur_tpdu_code<>CCCODE) then
                    sub_state := false;
                 else if (l_TSAP=0) then
                    l_TSAP:=get_sword(var_ptr.value,k);
                 else if (l_TSAP<>get_sword(var_ptr.value,k))) then
                    sub_state := false;

            PDUSIZCODE :         (* 1100 0000: TPDU size *)
                 if (cur_tpdu_code=CRCODE) then
                    size := get_sword (var_ptr.value, k);
                 else if (cur_tpdu_code<>CCCODE) then
                    sub_state := false;
                 else
                    begin
                    sn := get_sword(var_ptr.value,k);
                    if ((size=0) or (size>sn)) then
                         size := sn;
                    end;
```

```
VERSNOCODE :        (* 1100 0100: version number=1 *)
   vers := get_sword(var_ptr.value,k);

SECURICODE :        (* 1100 0101: protection *)
   prot := d_gets(var_ptr.value,1,k);

ADDLOPCODE :        (* 1100 0110: additional options *)
   begin
   sn := get_sword(var_ptr.value, k);
   if (sn and 2=0) then
        ch_flag := true;
     else
        ch_flag := false;
   if (sn and 1) then
        ed_flag := true;
     else
        ed_flag := false;
   end;

ALTCLSCODE :        (* 1100 0111: alternative class *)
   begin
   if (class<>4) then
        for i := 1 to k do
             if (var_ptr.value[i]=4) then
                  class := 4;
   if (class<>4) then
        sub_state := false;
   end;

AKTIMECODE :        (* 1000 0101: acknowledge time *)
   ack_time := d_gets(var_ptr.value,1,k);

THRUPTCODE :        (* 1000 1001: throughput *)
   throuput.max_AB := get_word(var_ptr.value[1],3);
   throuput.min_AB := get_word(var_ptr.value[4],3);
   throuput.max_BA := get_word(var_ptr.value[7],3);
   throuput.min_BA := get_word(var_ptr.value[10],3);
   if (k=24) then
        begin
        throuput.ave_AB := get_word(var_ptr.value[13],3);
        throuput.a_min_AB := get_word(var_ptr.value[16],3);
        throuput.ave_BA := get_word(var_ptr.value[19],3);
        throuput.a_min_BA := get_word(var_ptr.value[22],3);
        end;

RESERRCODE :        (* 1000 0110: residual error rate *)
   res_error.target := get_word(var_ptr.value[1],1);
   res_error.min_acc := get_word(var_ptr.value[2],1);
   res_error.TSDU_size := get_word(var_ptr.value[3],1);

PRIORTCODE :        (* 1000 0111: priority *)
   priority := get_word (var_ptr.value,k);

TRNDELCODE :        (* 1000 1000: transit delay *)
```

```
                trans_delay.target_AB := get_word(var_ptr.value[1],2);
                trans_delay.max_AB := get_word(var_ptr.value[3],2);
                trans_delay.target_BA := get_word(var_ptr.value[5],2);
                trans_delay.max_BA := get_word(var_ptr.value[7],2);

             default :                  (* CHKSUMCODE,REASSIGN,etc *)
                null;
             end; (* case *)

          var_ptr := ^var_ptr.value[k+1];
          l := l-k-2;
          end; (* while *)
      end;



  procedure accept_TP_mes(mes: TPDU_message_type);

      (* accept the control info in the TPDU from TCP *)

    var
      n: word_type;
      sn: short_word;
      i,j,k,l: integer;
      var_ptr: ^TP_var_part_type;

      begin
      tp_mes := mes;
      cur_tpdu_code := tp_mes.tpdu_code; (* 4 bits at right *)
      if (g_ref=0) then
          g_ref := tp_mes.src_ref;       (* fill in remote ref *)
      sub_state := true;                 (* true for OK *)

      case cur_tpdu_code of              (* pick up special fields *)
       CRCODE,CCCODE :
          begin
          class := tp_mes.class;
          if (cur_tpdu_code=CRCODE) then
              begin
              g_module := tp_mes.module mod 16;
              d_ip_addr := tp_mes.d_IP_addr;
              s_ip_addr := tp_mes.s_IP_addr;
              end;
          end;
        DRCODE :
          reason := tp_mes.reason;
        ERCODE :
          cause := tp_mes.reason;
        default :                        (* DC *)
          null;
      end;

      var_ptr := ^tp_mes.var_part;
      l := v_length;                     (* length ot var. part *)
```

```
while (1>0) do
     begin
     k := var_ptr.length;
     case var_ptr.param_code of

       CALINGCODE :        (* 1100 0001: calling TSAP-ID *)
            if (cur_tpdu_code=CRCODE) then
               l_TSAP := get_sword (var_ptr.value, k);
            else if (cur_tpdu_code<>CCCODE) then
               sub_state := false;
            else if (l_TSAP=0) then
               l_TSAP:=get_sword(var_ptr.value,k);
            else if (l_TSAP<>get_sword(var_ptr.value,k))) then
               sub_state := false;

       CALLEDCODE :        (* 1100 0010: called TSAP-ID *)
            if (cur_tpdu_code=CRCODE) then
               r_TSAP := get_sword (var_ptr.value, k);
            else if (cur_tpdu_code<>CCCODE) then
               sub_state := false;
            else if (r_TSAP=0) then
               r_TSAP:=get_sword(var_ptr.value,k);
            else if (r_TSAP<>get_sword(var_ptr.value,k))) then
               sub_state := false;

       PDUSIZCODE :        (* 1100 0000: TPDU size *)
            if (cur_tpdu_code=CRCODE) then
               g_size := get_sword (var_ptr.value, k);
            else if (cur_tpdu_code<>CCCODE) then
               sub_state := false;
            else
               begin
               sn := get_sword(var_ptr.value,k);
               if ((g_size=0) or (g_size>sn)) then
                    g_size := sn;
               end;

       VERSNOCODE :        (* 1100 0100: version number=1 *)
          g_vers := get_sword(var_ptr.value,k);

       SECURICODE :        (* 1100 0101: protection *)
          g_prot := d_gets(var_ptr.value,1,k);

       ALTCLSCODE :        (* 1100 0111: alternative class *)
          begin
          if (g_class<>4) then
               for i := 1 to k do
                    if (var_ptr.value[i]=4) then
                         g_class := var_ptr.value[i];
          if (g_class<>4) then
               sub_state := false;
          end;

       AKTIMECODE :        (* 1000 0101: acknowledge time *)
          g_ack_time := d_gets(var_ptr.value,1,k);
```

```
          THRUPTCODE :        (* 1000 1001: throughput *)
             begin
             g_throuput.max_AB := get_word(var_ptr.value[1],3);
             g_throuput.min_AB := get_word(var_ptr.value[4],3);
             g_throuput.max_BA := get_word(var_ptr.value[7],3);
             g_throuput.min_BA := get_word(var_ptr.value[10],3);
             if (k=24) then
                begin
                g_throuput.ave_AB := get_word(var_ptr.value[13],3);
                g_throuput.a_min_AB := get_word(var_ptr.value[16],3);
                g_throuput.ave_BA := get_word(var_ptr.value[19],3);
                g_throuput.a_min_BA := get_word(var_ptr.value[22],3
                end;
             end;

          RESERRCODE :        (* 1000 0110: residual error rate *)
             begin
             g_res_error.target := get_word(var_ptr.value[1],1);
             g_res_error.min_acc := get_word(var_ptr.value[2],1);
             g_res_error.TSDU_size := get_word(var_ptr.value[3],1);
             end;

          PRIORTCODE :        (* 1000 0111: priority *)
             g_priority := get_word (var_ptr.value,k);

          TRNDELCODE :        (* 1000 1000: transit delay *)
             begin
             g_trans_delay.target_AB := get_word(var_ptr.value[1],2)
             g_trans_delay.max_AB := get_word(var_ptr.value[3],2);
             g_trans_delay.target_BA := get_word(var_ptr.value[5],2);
             g_trans_delay.max_BA := get_word(var_ptr.value[7],2);
             end;

          default:
             null;
          end;

       var_ptr := ^var_ptr.value[k+1];
       l := l-k-2;
       end;
end;
```

```
function acceptable_CR : boolean;

     (* check to see if CR/CC from ISO-TP4 is acceptable *)

  var
     ok: boolean;

     begin
     ok := sub_state;                 (* set by accept_TPDU *)

     if((ok) and (class<>CLASS_4)) then
          ok := false;

     if (ok=true) then
          ok := acceptable_param();       (* check if parameters
                                             are acceptable *)

     acceptable_CR := ok;
     end;


function encode_chksum : TP_var_part_type;

     (* make checksum entry *)

     var
          var_ptr: TP_var_part_type;

     begin
     var_ptr := d_create(4);
     var_ptr.param_code := CHKSUMCODE;
     var_ptr.length := 2;
     var_ptr.value := 0;
     encode_chksum := var_ptr;
     end;


function encode_subseq : TP_var_part_type;

     (* make sub-sequence entry *)

     var
          var_ptr: TP_var_part_type;

     begin
     var_ptr := d_create(4);
     var_ptr.param_code := SUBSEQCODE;
     var_ptr.length := 2;
     var_ptr.value := r_sub_seq;
     encode_subseq := var_ptr;
     end;
```

```
procedure build_DR (reason: reason_type;
                    mes: IP_TP_message_type);

     (* build DR for ISO-TP4, in cases of:
          CLOSED * CC;
          WFCC * CC_not_acceptable;
          WBCL * CC;
          AKWAIT * ER;
          OPEN * ER;
          CLOSING * CR, CC, EA;
          REF_WAIT * CC              *)

     begin
     mes_DR.module := (mes.module mod 16) * 16;
                               (* shift l_module to d_module,
                                  leave l_module blank *)
     mes_DR.message := 0;
     mes_DR.s_IP_addr := mes.d_IP_addr;
     mes_DR.d_IP_addr := mes.s_IP_addr;

     ptr_CR := ^mes.data;
     mes_DR.tpdu.length := CR_VAR-1;    (* length of DR TPDU *)
     mes_DR.tpdu.tpdu_code := DRCODE*16;
     mes_DR.tpdu.dst_ref := ptr_CR.src_ref;
     mes_DR.tpdu.src_ref := ptr_CR.dst_ref;
     mes_DR.tpdu.reason := reason;
     if (ch_flag) then
          begin
          mes_DR.tpdu.length := mes_DR.tpdu.length+4;
          d_append(mes_DR.tpdu, encode_chksum);
          set_checksum(mes_DR.tpdu, mes_DR.tpdu.length, CR_VAR+3);
          end;
     end;



procedure build_DC (mes: IP_TP_message_type);

     (* build DC for ISO-TP4 in response to:
          CLOSED, WFCCout, AKWAIT, OPEN, REFWAIT * DR *)

     begin
     mes_DC.module := (mes.module mod 16) * 16
                    + mes.module div 16;
                         (* exchange l_module & d_module *)
     mes_DC.message := 0;
     mes_DC.s_IP_addr := mes.d_IP_addr;
     mes_DC.d_IP_addr := mes.s_IP_addr;

     ptr_DR := ^mes.data;
```

```
        mes_DC.tpdu.length := DC_VAR-1;     (* length of DC TPDU *)
        mes_DC.tpdu.tpdu_code := DCCODE*16;
        mes_DC.tpdu.dst_ref := ptr_DR.src_ref;
        mes_DC.tpdu.src_ref := ptr_DR.dst_ref;
        if (ch_flag) then
             begin
             mes_DC.tpdu.length := mes_DC.tpdu.length+4;
             d_append(mes_DC.tpdu, encode_chksum());
             set_checksum(mes_DC.tpdu, mes_DC.tpdu.length, DC_VAR+3);
             end;
        end;



   procedure build_CC (mes: IP_TP_message_type);

        (* build CC for ISO-TP4 in response to: AKWAIT * CR *)

        begin
        mes_CC.module := (mes.module mod 16) * 16
                       + mes.module div 16;
                                (* exchange l_module & d_module *)
        mes_CC.message := 0;
        mes_CC.s_IP_addr := mes.d_IP_addr;
        mes_CC.d_IP_addr := mes.s_IP_addr;

        mes_CC.tpdu.length := CC_VAR-1;     (* length of CC TPDU *)
        rcv_cdt := get_cdt;
        mes_CC.tpdu.tpdu_code := CCCODE*16 + rcv_cdt;
        mes_CC.tpdu.dst_ref := r_ref;
        mes_CC.tpdu.src_ref := l_ref;
        mes_CC.tpdu.class := class;
        if (ch_flag) then
             begin
             mes_CC.tpdu.length := mes_CC.tpdu.length+4;
             d_append(mes_CC.tpdu, encode_chksum());
             set_checksum(mes_CC.tpdu, mes_CC.tpdu.length, CR_VAR+3);
             end;
        end;



   procedure make_DR (reason: reason_type);

        (* make up DR for ISO-TP4, in cases of:
             WFCC, WBCL, AKWAIT, OPEN * retrans_timer * count=max;
             OPEN * inact_timer *)

        begin
        mes_DR.module := l_module * 16; (* shift l_module to d_module,
                                           leave l_module blank *)
        mes_DR.message := 0;
        mes_DR.s_IP_addr := d_ip_addr;
```

```
    mes_DR.d_IP_addr := s_ip_addr;

    mes_DR.tpdu.length := CR_VAR-1;      (* length of DR TPDU *)
    mes_DR.tpdu.tpdu_code := DRCODE*16;
    mes_DR.tpdu.dst_ref := r_ref;
    mes_DR.tpdu.src_ref := l_ref;
    mes_DR.tpdu.reason := reason;
    if (ch_flag) then
         begin
         mes_DR.tpdu.length := mes_DR.tpdu.length+4;
         d_append(mes_DR.tpdu, encode_chksum);
         set_checksum(mes_DR.tpdu, mes_DR.tpdu.length, CR_VAR+3);
         end;
    end;



procedure append_flow;   primitive;

    (* append flow-control info onto the AK *)



procedure make_AK;

    (* make up AK for ISO-TP4, in cases of:
         OPEN * CC,
         OPEN * DT, ED,
         OPEN * timers *)

    begin

    update_window();            (* update window control info *)

    if (ex_flag=false) then
         begin
         mes_AK.module := l_module * 16; (* shift l_mod to d_mod,
                                           leave l_mod blank *)
         mes_AK.message := 0;
         mes_AK.s_IP_addr := d_ip_addr;
         mes_AK.d_IP_addr := s_ip_addr;

         mes_AK.tpdu.length := DT_VAR-1;     (* length of AK TPDU *)
         mes_AK.tpdu.tpdu_code := AKCODE*16 + rcv_cdt;
         mes_AK.tpdu.dst_ref := r_ref;
         if (r_ack_seq<rcv_seq) then
              begin
              r_ack_seq := rcv_seq;
              r_sub_seq := 0;
              mes_AK.tpdu.seq_no := r_ack_seq;
              end;
           else
              begin
              mes_AK.tpdu.seq_no := r_ack_seq;
```

```
                r_sub_seq := r_sub_seq+1;
                mes_AK.tpdu.length := mes_AK.tpdu.length+4;
                d_append(mes_AK.tpdu, encode_subseq);
                end;
           if (ch_flag) then
                begin
                mes_AK.tpdu.length := mes_AK.tpdu.length+4;
                d_append(mes_AK.tpdu, encode_chksum);
                set_checksum(mes_AK.tpdu, mes_AK.tpdu.length, DT_VAR+3)
                end;
           end;
        else  (* ex_flag=true *)
           begin
           mes_AKe.module := l_module * 16; (* shift l_mod to d_mod,
                                              leave l_mod blank *)
           mes_AKe.message := 0;
           mes_AKe.s_IP_addr := d_ip_addr;
           mes_AKe.d_IP_addr := s_ip_addr;

           mes_AKe.tpdu.length := DTe_VAR-1;    (* length of AKe TPDU *
           mes_AKe.tpdu.tpdu_code := AKeCODE*16;
           mes_AKe.tpdu.cdt := rcv_cdt;
           mes_AKe.tpdu.dst_ref := r_ref;
           if (ch_flag) then
                begin
                mes_AKe.tpdu.length := mes_AKe.tpdu.length+4;
                d_append(mes_AKe.tpdu, encode_chksum);
                set_checksum(mes_AKe.tpdu, mes_AKe.tpdu.length,
                                      DTe_VAR+3);
                end;
           if (r_ack_seq<rcv_seq) then
                begin
                r_ack_seq := rcv_seq;
                r_sub_seq := 0;
                mes_AKe.tpdu.seq_no := r_ack_seq;
                end;
             else
                begin
                mes_AKe.tpdu.seq_no := r_ack_seq;
                r_sub_seq := r_sub_seq+1;
                mes_AKe.tpdu.length := mes_AKe.tpdu.length+4;
                d_append(mes_AKe.tpdu, encode_subseq);
                end;
           end;
      end;
```

```
procedure make_EA;

    (* make up EA for ISO-TP4, in cases of:
        OPEN * CC,
        OPEN * DT, ED,
        OPEN * timers *)

    begin

    update_window();          (* update window control info *)

    if (ex_flag=false) then
        begin
        mes_EA.module := l_module * 16; (* shift l_mod to d_mod,
                                           leave l_mod blank *)
        mes_EA.message := 0;
        mes_EA.s_IP_addr := d_ip_addr;
        mes_EA.d_IP_addr := s_ip_addr;

        mes_EA.tpdu.length := DT_VAR-1;     (* length of EA TPDU *)
        mes_EA.tpdu.tpdu_code := EACODE*16 + rcv_cdt;
        mes_EA.tpdu.dst_ref := r_ref;
        if (r_ack_seq<rcv_seq) then
            begin
            r_ack_seq := rcv_seq;
            r_sub_seq := 0;
            mes_EA.tpdu.seq_no := r_ack_seq;
            end;
          else
            begin
            mes_EA.tpdu.seq_no := r_ack_seq;
            r_sub_seq := r_sub_seq+1;
            mes_EA.tpdu.length := mes_EA.tpdu.length+4;
            d_append(mes_EA.tpdu, encode_subseq);
            end;
        if (ch_flag) then
            begin
            mes_EA.tpdu.length := mes_EA.tpdu.length+4;
            d_append(mes_EA.tpdu, encode_chksum);
            set_checksum(mes_EA.tpdu, mes_EA.tpdu.length, DT_VAR+3);
            end;
        end;
      else  (* ex_flag=true *)
        begin
        mes_EAe.module := l_module * 16; (* shift l_mod to d_mod,
                                            leave l_mod blank *)
        mes_EAe.message := 0;
        mes_EAe.s_IP_addr := d_ip_addr;
        mes_EAe.d_IP_addr := s_ip_addr;

        mes_EAe.tpdu.length := DTe_VAR-1;     (* length of EAe TPDU *
        mes_EAe.tpdu.tpdu_code := EAeCODE*16;
        mes_EAe.tpdu.cdt := rcv_cdt;
        mes_EAe.tpdu.dst_ref := r_ref;
```

```
            if (ch_flag) then
                begin
                mes_EAe.tpdu.length := mes_EAe.tpdu.length+4;
                d_append(mes_EAe.tpdu, encode_chksum);
                set_checksum(mes_EAe.tpdu, mes_EAe.tpdu.length,
                                          DTe_VAR+3);
                end;
            if (r_ack_seq<rcv_seq) then
                begin
                r_ack_seq := rcv_seq;
                r_sub_seq := 0;
                mes_EAe.tpdu.seq_no := r_ack_seq;
                end;
              else
                begin
                mes_EAe.tpdu.seq_no := r_ack_seq;
                r_sub_seq := r_sub_seq+1;
                mes_EAe.tpdu.length := mes_EAe.tpdu.length+4;
                d_append(mes_EAe.tpdu, encode_subseq);
                end;
            end;
        end;



    procedure trans_CR;

        (* translate CR of ISO-TP4 to GW format, in case of
             CLOSED * CR  *)

        begin
        tp_mes.module := mes_CR.module;
        tp_mes.message := mes_CR.message;
        tp_mes.s_IP_addr := mes_CR.s_IP_addr;
        tp_mes.d_IP_addr := mes_CR.d_IP_addr;

        tp_mes.tpdu_code := mes_CR.tpdu.tpdu_code/16;
        tp_mes.dst_ref := g_ref;
        tp_mes.src_ref := l_ref;
        tp_mes.class := mes_CR.tpdu.class;
        tp_mes.v_length := mes_CR.tpdu.length - tpdu_fixed() +1;
        tp_mes.var_part := d_create(tp_mes.v_length);
        d_puts (tp_mes.var_part, 1, mes_CR.tpdu.var_part);
        tp_mes.d_length := d_length(mes_CR.tpdu.data);
        tp_mes.data := d_create(tp_mes.d_length);
        d_puts (tp_mes.data, 1, mes_CR.tpdu.data);
        end;
```

```
procedure trans_CC;

     (* translate CC of ISO-TP4 to GW format, in case of
          WFCCiso * CC  *)

     begin
     tp_mes.module := mes_CC.module;
     tp_mes.message := mes_CC.message;
     tp_mes.s_IP_addr := mes_CC.s_IP_addr;
     tp_mes.d_IP_addr := mes_CC.d_IP_addr;

     tp_mes.tpdu_code := mes_CC.tpdu.tpdu_code/16;
     tp_mes.dst_ref := g_ref;
     tp_mes.src_ref := l_ref;
     tp_mes.class := mes_CC.tpdu.class;
     tp_mes.v_length := mes_CC.tpdu.length - tpdu_fixed() +1;
     tp_mes.var_part := d_create(tp_mes.v_length);
     d_puts (tp_mes.var_part, 1, mes_CC.tpdu.var_part);
     tp_mes.d_length := d_length(mes_CC.tpdu.data);
     tp_mes.data := d_create(tp_mes.d_length);
     d_puts (tp_mes.data, 1, mes_CC.tpdu.data);
     end;


procedure trans_DR;

     (* translate DR of ISO-TP4 to GW format, in case of
          WFCCout, AKWAIT, OPEN * DR  *)

     begin
     tp_mes.module := mes_DR.module;
     tp_mes.message := mes_DR.message;
     tp_mes.s_IP_addr := mes_DR.s_IP_addr;
     tp_mes.d_IP_addr := mes_DR.d_IP_addr;

     tp_mes.tpdu_code := mes_DR.tpdu.tpdu_code/16;
     tp_mes.dst_ref := g_ref;
     tp_mes.src_ref := l_ref;
     tp_mes.reason := mes_DR.tpdu.reason;
     tp_mes.v_length := mes_DR.tpdu.length - tpdu_fixed() +1;
     tp_mes.var_part := d_create(tp_mes.v_length);
     d_puts (tp_mes.var_part, 1, mes_DR.tpdu.var_part);
     tp_mes.d_length := d_length(mes_DR.tpdu.data);
     tp_mes.data := d_create(tp_mes.d_length);
     d_puts (tp_mes.data, 1, mes_DR.tpdu.data);
     end;
```

```
procedure trans_DT;

     (* translate DT of ISO-TP4 to GW format, in case of
          AKWAITiso, OPEN * DT  *)

     begin
     tp_mes.dst_ref := g_ref;
     tp_mes.src_ref := l_ref;
     if (ex_flag=false) then
          begin
          tp_mes.module := mes_DT.module;
          tp_mes.message := mes_DT.message;
          tp_mes.s_IP_addr := mes_DT.s_IP_addr;
          tp_mes.d_IP_addr := mes_DT.d_IP_addr;
          tp_mes.tpdu_code := mes_DT.tpdu.tpdu_code/16;
          tp_mes.class := mes_DT.tpdu.class;
          tp_mes.v_length := mes_DT.tpdu.length - tpdu_fixed() +1;
          tp_mes.var_part := d_create(tp_mes.v_length);
          d_puts (tp_mes.var_part, 1, mes_DT.tpdu.var_part);
          tp_mes.d_length := d_length(mes_DT.tpdu.data);
          tp_mes.data := d_create(tp_mes.d_length);
          d_puts (tp_mes.data, 1, mes_DT.tpdu.data);
          end;
       else
          begin
          tp_mes.module := mes_DTe.module;
          tp_mes.message := mes_DTe.message;
          tp_mes.s_IP_addr := mes_DTe.s_IP_addr;
          tp_mes.d_IP_addr := mes_DTe.d_IP_addr;
          tp_mes.tpdu_code := mes_DTe.tpdu.tpdu_code/16;
          tp_mes.class := mes_DTe.tpdu.class;
          tp_mes.v_length := mes_DTe.tpdu.length - tpdu_fixed() +1;
          tp_mes.var_part := d_create(tp_mes.v_length);
          d_puts (tp_mes.var_part, 1, mes_DTe.tpdu.var_part);
          tp_mes.d_length := d_length(mes_DTe.tpdu.data);
          tp_mes.data := d_create(tp_mes.d_length);
          d_puts (tp_mes.data, 1, mes_DTe.tpdu.data);
          end;
     end;



procedure trans_ED;

     (* translate ED of ISO-TP4 to GW format, in case of
          AKWAITiso, OPEN * ED  *)

     begin
     tp_mes.dst_ref := g_ref;
     tp_mes.src_ref := l_ref;
     if (ex_flag=false) then
          begin
          tp_mes.module := mes_ED.module;
```

```
        tp_mes.message := mes_ED.message;
        tp_mes.s_IP_addr := mes_ED.s_IP_addr;
        tp_mes.d_IP_addr := mes_ED.d_IP_addr;
        tp_mes.tpdu_code := mes_ED.tpdu.tpdu_code/16;
        tp_mes.class := mes_ED.tpdu.class;
        tp_mes.v_length := mes_ED.tpdu.length - tpdu_fixed() +1;
        tp_mes.var_part := d_create(tp_mes.v_length);
        d_puts (tp_mes.var_part, 1, mes_ED.tpdu.var_part);
        tp_mes.d_length := d_length(mes_ED.tpdu.data);
        tp_mes.data := d_create(tp_mes.d_length);
        d_puts (tp_mes.data, 1, mes_ED.tpdu.data);
        end;
     else
        begin
        tp_mes.module := mes_EDe.module;
        tp_mes.message := mes_EDe.message;
        tp_mes.s_IP_addr := mes_EDe.s_IP_addr;
        tp_mes.d_IP_addr := mes_EDe.d_IP_addr;
        tp_mes.tpdu_code := mes_EDe.tpdu.tpdu_code/16;
        tp_mes.class := mes_EDe.tpdu.class;
        tp_mes.v_length := mes_EDe.tpdu.length - tpdu_fixed() +1;
        tp_mes.var_part := d_create(tp_mes.v_length);
        d_puts (tp_mes.var_part, 1, mes_EDe.tpdu.var_part);
        tp_mes.d_length := d_length(mes_EDe.tpdu.data);
        tp_mes.data := d_create(tp_mes.d_length);
        d_puts (tp_mes.data, 1, mes_EDe.tpdu.data);
        end;
   end;


procedure make_DRg (reason);

     (* makeup DRg in GW format, in case of
         WFCCiso * CC not acceptable,
         WFCCiso, AKWAIT, OPEN * ER,
         retrans_timer & count=max,
         inact_timer *)

     begin
     tp_mes.module := g_module * 16;
     tp_mes.message := 0;
     tp_mes.s_IP_addr := s_ip_addr;
     tp_mes.d_IP_addr := d_ip_addr;

     tp_mes.tpdu_code := DRCODE;
     tp_mes.dst_ref := g_ref;
     tp_mes.src_ref := l_ref;
     tp_mes.reason := reason;
     tp_mes.v_length := 0;
     tp_mes.d_length := 0;
     end;
```

```
procedure trans_CRg;

     (* translate CR of GW format to ISO-TP4 format, in case of
          CLOSED * CRg  *)

     begin
     mes_CR.module := tp_mes.module;
     mes_CR.message := tp_mes.message;
     mes_CR.s_IP_addr := tp_mes.s_IP_addr;
     mes_CR.d_IP_addr := tp_mes.d_IP_addr;

     mes_CR.tpdu.tpdu_code := tp_mes.tpdu_code * 16;
     mes_CR.tpdu.dst_ref := r_ref;
     mes_CR.tpdu.src_ref := l_ref;
     mes_CR.tpdu.class := tp_mes.class;
     tp_mes.var_part := d_create(tp_mes.v_length);
     d_puts (mes_CR.tpdu.var_part, 1, tp_mes.var_part);
     mes_CR.tpdu.data := d_create(tp_mes.d_length);
     d_puts (mes_CR.tpdu.data, 1, tp_mes.data);
     end;


procedure trans_CCg;

     (* translate CC of GW format to ISO-TP4 format, in case of
          WFCCout * CCg  *)

     begin
     mes_CC.module := tp_mes.module;
     mes_CC.message := tp_mes.message;
     mes_CC.s_IP_addr := tp_mes.s_IP_addr;
     mes_CC.d_IP_addr := tp_mes.d_IP_addr;

     mes_CC.tpdu.tpdu_code := tp_mes.tpdu_code * 16;
     mes_CC.tpdu.dst_ref := r_ref;
     mes_CC.tpdu.src_ref := l_ref;
     mes_CC.tpdu.class := tp_mes.class;
     tp_mes.var_part := d_create(tp_mes.v_length);
     d_puts (mes_CC.tpdu.var_part, 1, tp_mes.var_part);
     mes_CC.tpdu.data := d_create(tp_mes.d_length);
     d_puts (mes_CC.tpdu.data, 1, tp_mes.data);
     end;
```

```
procedure trans_DRg;

     (* translate DR of GW format to ISO-TP4 format, in case of
          WFCCiso, WFCCout, AKWAITiso, OPEN * DRg  *)

     begin
     mes_DR.module := tp_mes.module;
     mes_DR.message := tp_mes.message;
     mes_DR.s_IP_addr := tp_mes.s_IP_addr;
     mes_DR.d_IP_addr := tp_mes.d_IP_addr;

     mes_DR.tpdu.tpdu_code := tp_mes.tpdu_code * 16;
     mes_DR.tpdu.dst_ref := r_ref;
     mes_DR.tpdu.src_ref := l_ref;
     mes_DR.tpdu.reason := tp_mes.reason;
     tp_mes.var_part := d_create(tp_mes.v_length);
     d_puts (mes_DR.tpdu.var_part, 1, tp_mes.var_part);
     mes_DR.tpdu.data := d_create(tp_mes.d_length);
     d_puts (mes_DR.tpdu.data, 1, tp_mes.data);
     end;



procedure trans_DTg;

     (* translate DT of GW format to ISO-TP4 format, in case of
          AKWAITout, OPEN * DTg  *)

     begin
     if (ex_flag=false) then
          begin
          mes_DT.module := tp_mes.module;
          mes_DT.message := tp_mes.message;
          mes_DT.s_IP_addr := tp_mes.s_IP_addr;
          mes_DT.d_IP_addr := tp_mes.d_IP_addr;

          mes_DT.tpdu.tpdu_code := tp_mes.tpdu_code * 16;
          mes_DT.tpdu.dst_ref := r_ref;
          mes_DT.tpdu.src_ref := l_ref;
          mes_DT.tpdu.class := tp_mes.class;
          tp_mes.var_part := d_create(tp_mes.v_length);
          d_puts (mes_DT.tpdu.var_part, 1, tp_mes.var_part);
          mes_DT.tpdu.data := d_create(tp_mes.d_length);
          d_puts (mes_DT.tpdu.data, 1, tp_mes.data);
          end;
       else  (* ex_flag=true *)
          begin
          mes_DTe.module := tp_mes.module;
          mes_DTe.message := tp_mes.message;
          mes_DTe.s_IP_addr := tp_mes.s_IP_addr;
          mes_DTe.d_IP_addr := tp_mes.d_IP_addr;

          mes_DTe.tpdu.tpdu_code := tp_mes.tpdu_code * 16;
```

```
            mes_DTe.tpdu.dst_ref := r_ref;
            mes_DTe.tpdu.src_ref := l_ref;
            mes_DTe.tpdu.class := tp_mes.class;
            tp_mes.var_part := d_create(tp_mes.v_length);
            d_puts (mes_DTe.tpdu.var_part, 1, tp_mes.var_part);
            mes_DTe.tpdu.data := d_create(tp_mes.d_length);
            d_puts (mes_DTe.tpdu.data, 1, tp_mes.data);
            end;
        end;



    procedure trans_EDg;

        (* translate ED of GW format to ISO-TP4 format, in case of
            AKWAITout, OPEN * EDg  *)

        begin
        if (ex_flag=false) then
            begin
            mes_ED.module := tp_mes.module;
            mes_ED.message := tp_mes.message;
            mes_ED.s_IP_addr := tp_mes.s_IP_addr;
            mes_ED.d_IP_addr := tp_mes.d_IP_addr;

            mes_ED.tpdu.tpdu_code := tp_mes.tpdu_code * 16;
            mes_ED.tpdu.dst_ref := r_ref;
            mes_ED.tpdu.src_ref := l_ref;
            mes_ED.tpdu.class := tp_mes.class;
            tp_mes.var_part := d_create(tp_mes.v_length);
            d_puts (mes_ED.tpdu.var_part, 1, tp_mes.var_part);
            mes_ED.tpdu.data := d_create(tp_mes.d_length);
            d_puts (mes_ED.tpdu.data, 1, tp_mes.data);
            end;
          else  (* ex_flag=true *)
            begin
            mes_EDe.module := tp_mes.module;
            mes_EDe.message := tp_mes.message;
            mes_EDe.s_IP_addr := tp_mes.s_IP_addr;
            mes_EDe.d_IP_addr := tp_mes.d_IP_addr;

            mes_EDe.tpdu.tpdu_code := tp_mes.tpdu_code * 16;
            mes_EDe.tpdu.dst_ref := r_ref;
            mes_EDe.tpdu.src_ref := l_ref;
            mes_EDe.tpdu.class := tp_mes.class;
            tp_mes.var_part := d_create(tp_mes.v_length);
            d_puts (mes_EDe.tpdu.var_part, 1, tp_mes.var_part);
            mes_EDe.tpdu.data := d_create(tp_mes.d_length);
            d_puts (mes_EDe.tpdu.data, 1, tp_mes.data);
            end;
        end;
```

```
procedure check_queue;   primitive;

     (* Check all the TPDUs in the re-trans queue for re-
          transmission if necessary.
        As specified in IS 8073, it is implementation-dependent
          to have timer control over the 1st TPDU only and re-
          transmit all TPDUs in the queue on time-out; or to have
          timer control over all the single TPDUs in the re-trans
          queue. *)
```

```
(***                                               ***)
(***              Initialization                   ***)
(***                                               ***)


initialize

    state := CLOSED;
    proto_err := 0;

    exp_flag := false;
    exp_seq := 0;
    exp_que.last := 0;



(***                                               ***)
(***      Transitions of TP4_machine               ***)
(***                                               ***)


(***   GROUP-1:  TPDU from ISO-TP4     ***)

(***   State_transition from  CLOSED   ***)


trans
  when TP4M_I.TP4M_MESind                      (* CR from TP4 *)
    provided ((TP4_code(message)=CRCODE)) and (l_ref=0))
      from CLOSED to same

    begin                                  (* no conn available *)
    build_DR (R_peer_congestion, message);
    output TP4M_O.TP4M_MESreq(mes_DR);
    end;
```

```
trans
  when TP4M_I.TP4M_MESind                          (* CR from TP4 *)
    provided ((TP4_code(message)=CRCODE)) and (l_ref<>0))
      from CLOSED

     begin
     accept_TPDU (message);             (* take in the TPDU *)
     accept_TP4_param();
     if (acceptable_CR=false) then
          begin
          build_DR (R_CR_refused, message);
          output TP4M_O.TP4M_MESreq (mes_DR);
          next__state := same;
          end;
       else
          begin
          send_seq := 0;
          send_cdt := cdt;
          s_sub_seq := 0;
          s_ack_seq := 0;

     (*   increase_count;    *)
     (*   set_retrans_timer; *)
       (* These two actions are specified in the original IS
          8073. But there might be some mistakes in it, because
          there is no specification in the CLASS-4 state table
          for the following state (WFTRESP) with the event of
          retrans_timer, and for the case of count exceeding the
          limit. It is assumed here that the CR_receiving side
          will passively wait for the TCONind from TP-4 user, or
          CRg from the TP-GW. If either the TCONind or CRg never
          comes, the calling TP-4 will finally timeout on the CR,
          and will send DR to terminate the connection
          establishment process. *)

          rcv_seq := 0;
          rcv_cdt := set_credit;
          r_sub_seq := 0;
          r_ack_seq := 0;
          trans_CR;
          output TP4I_O.TP4I_MESreq(tp_mes); (* send translated
                                                message to linker *)
          next_state := WFCCout;
          end;
     end;
```

```
trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = CCCODE)
      from CLOSED to SAME

     begin
     build_DR(R_ref_mismatched, message);
     output TP4M_O.TP4M_MESreq(mes_DR);
     end;



trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = DRCODE)
     from CLOSED to SAME

     begin
     ptr_DR := ^message.data;
     if(ptr_DR.src_ref<>0) then
          build_DC(message);
          output TP4M_O.TP4M_MESreq(mes_DC);
     end;


trans
  when TP4M_I.TP4M_MESind
    provided ((TP4_code(message) = DTCODE) or
              (TP4_code(message) = AKCODE) or
              (TP4_code(message) = EDCODE) or
              (TP4_code(message) = EACODE) or
              (TP4_code(message) = DCCODE) or
              (TP4_code(message) = ERCODE) )
     from CLOSED to SAME

     begin
     proto_err := proto_err+1;
     end;
```

```
(***   State_transition from  WFCCiso   ***)


trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = CCCODE)
      from WFCCiso

     var
          retrans_ptr := ^retrans_type;

     begin
     stop_retrans_timer;                  (* for CR *)
     accept_TPDU (message);               (* take in the TPDU *)
     accept_TP4_param();
     if (acceptable_CR()) then
          begin
          send_seq := 0;
          send_cdt := cdt;
          s_sub_seq := 0;
          s_ack_seq := 0;

          re_trans.total := re_trans.total-1;  (* remove CR from
                                                  retrans_queue *)
          set_window_timer;
          set_inact_timer;
          trans_CC;
          output TP4I_O.TP4I_MESreq(tp_mes); (* send translated
                                              message to linker *)
          next_state := AKWAITout;
          end;
      else                                  (* CC not acceptable *)
          begin
          build_DR(R_negotiation_failed, message);
          output TP4M_O.TP4M_MESreq(mes_DR);
          retrans_ptr := re_trans.buf[1]; (* replace CR with DR *)
          retrans_ptr.data := mes_DR;
          retrans_ptr.count := 1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          make_DRg;
          output TP4I_O.TP4I_MESreq(tp_mes);  (* send TDISind *)
          next_state := CLOSING;
          end;
     end;
```

```
trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = DRCODE)
      from WFCCiso to REFWAIT

     begin
     accept_TPDU (message);          (* take in the TPDU *)
     accept_TP4_param();             (* but DC is not needed *)
     set_ref_timer;
     trans_DR;
     output TP4I_O.TP4I_MESreq(tp_mes);  (* send TDISind to linker *)
     end;



trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = ERCODE)
      from WFCCiso to REFWAIT

     begin
     accept_TPDU (message);            (* take in the TPDU *)
     accept_TP4_param();
     set_ref_timer;
     make_DRg;
     output TP4I_O.TP4I_MESreq(tp_mes);  (* send TDISind to linker *)
     end;
```

```
(***   State_transition from  WBCL   ***)


trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = CCCODE)
      from WBCL to CLOSING

     var
          retrans_ptr : retrans_type;

     begin
     stop_retrans_timer;
     accept_TPDU (message);          (* take in the TPDU *)
     accept_TP4_param();
     build_DR(R_negotiation_failed, message);
     output TP4M_O.TP4M_MESreq(mes_DR);
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := mes_DR;   (* replace CR *)
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     end;



trans
  when TP4M_I.TP4M_MESind
    provided ((TP4_code(message) = DRCODE) or
              (TP4_code(message) = ERCODE) )
      from WBCL to REFWAIT

     begin
     set_ref_timer;
     end;
```

```
(***   State_transition from  WFCCout    ***)


trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = CRCODE)
      from WFCCout to SAME

     begin
     null;                             (* waiting for TCONresp *)
     end;



trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = DRCODE)
     from WFCCout to CLOSED

     begin
     accept_TPDU (message);                 (* take in the TPDU *)
     accept_TP4_param();
     build_DC(message);
     output TP4M_O.TP4M_MESreq(mes_DC);
     trans_DR;
     output TP4I_O.TP4I_MESreq(tp_mes); (* send TDISind to linker *)
     flag_kill_me := now;                   (* terminate the TPM *)
     end;
```

```
(***   State_transition from  AKWAIT   ***)


trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = CRCODE)
      from AKWAIT to SAME

     var
          retrans_ptr: retrans_type;

     begin
     stop_retrans_timer;
     retrans_ptr := re_trans.buf[1];
     output TP4M_O.TP4M_MESreq(retrans_ptr.data);  (* retrans CC *)
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     end;



trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = ERCODE)
     from AKWAIT to CLOSING

     var
          retrans_ptr: retrans_type;

     begin
     stop_retrans_timer;
     accept_TPDU (message);           (* take in the TPDU *)
     accept_TP4_param();
     retrans_ptr := re_trans.buf[1];
     build_DR(R_protocol_error, message);
     output TP4M_O.TP4M_MESreq(mes_DR);
     retrans_ptr.data := mes_DR;         (* replace CC with DR *)
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     make_DRg;
     output TP4I_O.TP4I_MESreq(tp_mes);  (* send TDISind *)
     end;
```

```
trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = DRCODE)
      from AKWAIT to REFWAIT

     begin
     accept_TPDU (message);          (* take in the TPDU *)
     accept_TP4_param();
     build_DC(message);
     output TP4M_O.TP4M_MESreq(mes_DC);
     set_ref_timer;
     trans_DR;
     output TP4I_O.TP4I_MESreq(tp_mes);  (* send TDISind *)
     end;
```

```
(***   State_transition from  AKWAITiso   ***)

trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = DTCODE)
     from AKWAITiso to OPEN

   var
     out_flag: logical;

     begin
     stop_retrans_timer;                    (* no more re-trans CC *)
     re_trans.last := 0;                    (* clear re-trans queue *)

     accept_TPDU (message);                 (* take in the TPDU *)
     accept_TP4_param();
     out_flag := false;
     if ((TP4_code(message)=DTCODE) and (acceptable_DT=true)) then
          begin
          trans_DT;
          out_flag := true;
          end;
      else if((TP4_code(message)=EDCODE) and (acceptable_ED=true))
        then
          begin
          trans_ED;
          out_flag := true;
          end;
      else if((TP4_code(message)=AKCODE) and (acceptable_AK=true))
        then
          begin
          trans_AK;
          out_flag := true;
          end;
     if (out_flag=true) then
          output TP4I_O.TP4I_MESreq(tp_mes);
     set_inact_timer;
     set_inact_ack_timer;               (* for sending AK at suitable
                                           interval in absence of
                                           DT or ED *)

     end;
```

```
(***   State_transition from  OPEN  ***)

trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = CRCODE)
      from OPEN to SAME

     begin
     stop_inact_timer;
     set_inact_timer;
     end;



trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = CCCODE)
      from OPEN to SAME

     begin                                          (* duplicated CC *)
     stop_inact_timer;
     make_AK;
     output TP4M_O.TP4M_MESreq (mes_AK);            (* repeat AK *)
     set_inact_timer;
     end;


trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = ERCODE)
     from OPEN to CLOSING

     var
          retrans_ptr: retrans_type;

     begin
     stop_window_timer;
     stop_inact_timer;
     stop_retrans_timer;
     accept_TPDU (message);          (* take in the TPDU *)
     accept_TP4_param();
     build_DR(R_protocol_error, message);
     output TP4M_O.TP4M_MESreq(mes_DR);
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := mes_DR;
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     make_DRg;
     output TP4I_O.TP4I_MESreq(tp_mes);  (* send TDISind to linker *)
     end;
```

```
trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = DRCODE)
      from OPEN to REFWAIT

     var
          retrans_ptr: retrans_type;

     begin
     stop_retrans_timer;
     accept_TPDU (message);          (* take in the TPDU *)
     accept_TP4_param();
     build_DC(message);
     output TP4M_O.TP4M_MESreq(mes_DC);
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := mes_DC;
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     trans_DR;
     output TP4I_O.TP4I_MESreq(tp_mes);  (* send TDISind to linker *)
     set_ref_timer;
     end;


trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = DTCODE)
     from OPEN to SAME

     begin
     stop_inact_timer;
     accept_TPDU (message);             (* take in the TPDU *)
     accept_TP4_param();

     if (seq = rcv_seq) then            (* DT in sequence *)
          begin
          trans_DT;
          output TP4I_O.TP4I_MESreq(tp_mes);
          set_data_ack_timer;           (* set timer to ack
                                               during A_1 *)
          rcv_seq := rcv_seq + 1;
          while ((re_order.last>0) and   (* check re-seq queue *)
              (rcv_seq=re_order.buf[1].seq)) do
               begin
               message := re_order.buf[1].data;
               trans_DT;
               output TP4I_O.TP4I_MESreq(tp_mes);
               rcv_seq := rcv_seq + 1;
```

```
                for i:=2 to re_order.last do
                     re_order.buf[i-1] := re_order.buf[i];
                re_order.last := re_order.last -1;
           end;
       else if(((seq<rcv_seq) and (seq>rcv_seq-MAX_CDT)) or
               ((seq>rcv_seq+rcv_cdt) and
                (seq<rcv_seq+rcv_cdt+MAX_CDT))) then
                               (* out of window, but whthin limit:
                                    AKed with flow ctrl info *)
           begin
           make_AK;
           append_flow;
           output TP4M_O.TP4M_MESreq(mes_AK);
           end;
       else
           begin                         (* not-in-seq handling *)
           j := 0;
           for i := 1 to re_order.last do
             if ((seq>re_order.buf[i].seq) and (j=0)) then
                j := i;
           for i:=j to re_order.last do
                re_order.buf[i+1] := re_order.buf[i];
           re_order.buf[i].seq := seq;
           re_order.buf[i].data := message;   (* queued *)
           end;
      set_inact_timer;
      end;



trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = AKCODE)
     from OPEN to SAME

     begin
     stop_inact_timer;
     stop_retrans_timer;
     accept_TPDU (message);            (* take in the TPDU *)
     accept_TP4_param();

     if ((seq > s_ack_seq) or            (* AK in sequence *)
         ((seq=s_ack_seq) and (sub_seq>s_sub_seq))) then
          begin
          s_ack_seq := seq;
          s_sub_seq := sub_seq;
          while ((re_trans.last>0) and   (* check re-seq queue *)
              (s_ack_seq>=re_trans.buf[1].seq)) do
               begin
               for i:=2 to re_trans.last do
                    re_trans.buf[i-1] := re_trans.buf[i];
               re_trans.last := re_trans.last -1;
          end;
```

```
                                        (* else, automatic drop the AK *)
      if (re_trans.last>0) then
           set_retrans_timer;
      set_inact_timer;
      end;




trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = EDCODE)
     from OPEN to SAME

   var
     out_flag: logical;

     begin
     stop_inact_timer;
     accept_TPDU (message);                    (* take in the TPDU *)
     accept_TP4_param();
     trans_ED;
     output TP4I_O.TP4I_MESreq(tp_mes);
     build_EA;
     output TP4M_O.TP4M_MESreq(mes_EA);
     re_trans.last := re_trans.last + 1;
     retrans_ptr := re_trans.buf[re_trans.last];
     retrans_ptr.data := mes_EA;
     retrans_ptr.count := 0;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     set_inact_timer;
     end;




trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = EACODE)
     from OPEN to SAME

     begin
     stop_inact_timer;
     stop_retrans_timer;
     accept_TPDU (message);                    (* take in the TPDU *)
     accept_TP4_param();

     if (seq >= exp_seq) then                  (* EA in sequence *)
          begin
          exp_seq := 0;
          exp_flag := false;                   (* OK for DT to go now *)
```

```
        while (exp_que.last>0) do      (* check queued DT due to
                                          ED blocking *)
            begin
            message := exp_que.buf[1];
            trans_DTg;
            output TP4M_O.TP4M_MESreq(mes_DT);
            re_trans.last := re_trans.last + 1;
            retrans_ptr := re_trans.buf[re_trans.last];
            retrans_ptr.data := mes_EA;
            retrans_ptr.count := 0;
            retrans_ptr.timer := RETRANS_TIME;
            for i:=2 to exp_que.last do
                exp_que.buf[i-1] := exp_que.buf[i];
            exp_que.last := exp_que.last -1;
        end;
    if (exp_que.last>0) then
        set_retrans_timer;
    set_inact_timer;
    end;
```

```
(***   State_transition from  CLOSING   ***)

trans
  when TP4M_I.TP4M_MESind
    provided ((TP4_code(message) = CRCODE) or
              (TP4_code(message) = CCCODE) or
              (TP4_code(message) = DTCODE) or
              (TP4_code(message) = AKCODE) or
              (TP4_code(message) = EDCODE) or
              (TP4_code(message) = EACODE) )
      from CLOSING to SAME

  var
     reason: reason_type;

     begin
     accept_TPDU (message);          (* take in the TPDU *)
     accept_TP4_param();
     if (TP4_code(message)=CRCODE) then
          reason := R_CR_refused;
      else if (TP4_code(message) = CCCODE) then
          reason := R_negotiation_failed;
      else
          reason := R_protocol_error;
     build_DR(reason, message);
     output TP4M_O.TP4M_MESreq(mes_DR);
     end;



trans
  when TP4M_I.TP4M_MESind
    provided ((TP4_code(message) = DRCODE) or
              (TP4_code(message) = DCCODE) or
              (TP4_code(message) = ERCODE) )
      from CLOSING to REFWAIT

     begin
     set_ref_timer;
     end;
```

```
(***   State_transition from  REFWAIT   ***)


trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = CCCODE)
      from REFWAIT to SAME

     begin
     accept_TPDU (message);          (* take in the TPDU *)
     accept_TP4_param();
     build_DR(R_protocol_error, message);
     output TP4M_O.TP4M_MESreq(mes_DR);
     end;



trans
  when TP4M_I.TP4M_MESind
    provided (TP4_code(message) = DRCODE)
      from REFWAIT to SAME

     begin
     accept_TPDU (message);          (* take in the TPDU *)
     accept_TP4_param();
     if (r_ref<>0) then
          build_DC(message);
          output TP4M_O.TP4M_MESreq(mes_DC);
     end;



trans
  when TP4M_I.TP4M_MESind
    provided ((TP4_code(message) = DTCODE) or
              (TP4_code(message) = AKCODE) or
              (TP4_code(message) = EDCODE) or
              (TP4_code(message) = EACODE) or
              (TP4_code(message) = DCCODE) or
              (TP4_code(message) = ERCODE) )
     from REFWAIT to SAME

     begin
     null;                   (* waiting for ref_timer to expire *)
     end;
```

```
(***   GROUP-2:  timer-related   ***)


trans
  from WFCCiso
    provided (timer(retrans))

     var
          retrans_ptr: ^retrans_type;

     begin                             (* only CR in retrans queue *)
     retrans_ptr := re_trans.buf[1];
     if (retrans_ptr.count < RETRANS_MAX) then
          begin
          output TP4M_O.TP4M_MESreq(mes_CR); (* re-trans CR *)
          retrans_ptr.count := retrans_ptr.count+1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          next_state := SAME;
          end;
      else begin                                (* count = max *)
          stop_window_timer;
          stop_inact_timer;
          if (local_choice=false) then
               begin
               make_DR(R_retrans_timer);     (* unspecified ! *)
               output TP4M_O.TP4M_MESreq(mes_DR);
               retrans_ptr.data := mes_DR;
               retrans_ptr.count := 1;
               retrans_ptr.timer := RETRANS_TIME;
               set_retrans_timer;
               make_DRg(R_retrans_timer);
               output TP4I_O.TP4I_MESreq(tp_mes); (* send TDISind *)
               next_state := CLOSING;
               end;
           else                                 (* local choice *)
               begin
               make_DRg(R_retrans_timer);
               output TP4I_O.TP4I_MESreq(tp_mes); (* send TDISind *)
               set_ref_timer;
               next_state := REFWAIT;
               end;
          end; (* else count=max *)
     end;
```

```
trans
  provided (timer(retrans))
    from WBCL

     var
          retrans_ptr: retrans_type;

     begin
     retrans_ptr := re_trans.buf[1];          (* only CR in queue *)
     if (retrans_ptr.count < RETRANS_MAX) then
          begin
          output TP4M_O.TP4M_MESreq(mes_CR);       (* re-trans *)
          retrans_ptr.count := retrans_ptr.count+1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          next_state := SAME;
          end;
      else                                     (* count = max *)
          begin
          stop_window_timer;
          stop_inact_timer;
          if (local_choice=false) then
               begin
               make_DR(R_retrans_timer);
               output TP4M_O.TP4M_MESreq(mes_DR);
               retrans_ptr.data := mes_DR;   (* replace CR *)
               retrans_ptr.count := 1;
               retrans_ptr.timer := RETRANS_TIME;
               set_retrans_timer;
               next_state := CLOSING;
               end;
           else                           (* not local choice *)
               begin
               set_ref_timer;
               next_state := REFWAIT;
               end;
          end;  (* count = max *)
      end;
```

```
trans
  provided (timer(retrans))
     from AKWAIT

     var
          retrans_ptr: retrans_type;

     begin
     retrans_ptr := re_trans.buf[1];
     if(retrans_ptr.count < RETRANS_MAX) then
          begin
          output TP4M_O.TP4M_MESreq(mes_CC);   (* retrans CC *)
          retrans_ptr.count := retrans_ptr.count+1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          next_state := SAME;
          end;
      else                                    (* count = max *)
          begin
          stop_window_timer;
          stop_inact_timer;
          make_DR(R_retrans_timer);
          output TP4M_O.TP4M_MESreq(mes_DR);
          retrans_ptr.data := mes_DR;    (* replace CC with DR *)
          retrans_ptr.count := 1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          make_DRg(R_retrans_timer);
          output TP4I_O.TP4I_MESreq(tp_mes);  (* send TDISind *)
          next_state := CLOSING;
          end;
     end;


trans
  from OPEN
    provided (timer(retrans))

     var
          retrans_ptr: retrans_type;

     begin
     retrans_ptr := re_trans.buf[1];
     if (retrans_ptr.count < RETRANS_MAX) then
          begin
          output TP4M_O.TP4M_MESreq(retrans_ptr.data);
          retrans_ptr.count := retrans_ptr.count+1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          check_queue;                  (* check for re-trans *)
          next_state := SAME;
          end;
```

```
          else begin                              (* count=MAX *)
             stop_window_timer;
             stop_inact_timer;
             re_trans.total := 1;           (* remove all retrans *)
             retrans_ptr := re_trans.buf[1];
             make_DR(R_retrans_timer);
             output TP4M_O.TP4M_MESreq(mes_DR);
             retrans_ptr.data := mes_DR;
             retrans_ptr.count := 1;
             retrans_ptr.timer := RETRANS_TIME;
             set_retrans_timer;
             make_DRg(R_retrans_timer);
             output TP4I_O.TP4I_MESreq(tp_mes); (* send TDISind *)
             next_state := CLOSING;
        end;



   trans
     provided (timer(retrans))
        from CLOSING

        var
             retrans_ptr: retrans_type;

        begin
        retrans_ptr := re_trans.buf[1];
        if (retrans_ptr.count < RETRANS_MAX) then
             begin
             output TP4M_O.TP4M_MESreq(mes_DR);
             retrans_ptr.count := retrans_ptr.count+1;
             retrans_ptr.timer := RETRANS_TIME;
             set_retrans_timer;
             next_state := SAME;
             end;
         else                                        (* count = MAX *)
             begin
             set_ref_timer;
             next_state := REFWAIT;
             end;
        end;
```

```
trans
  provided (timer(window))
    from OPEN to SAME            (* need to update window info *)

     var
          retrans_ptr: retrans_type;

     begin
     rcv_cdt := set_credit;      (* get new credit *)
     make_AK;
     append_flow;                (* append flow-ctrl parameters *)
     output TP4M_O.TP4M_MESreq(mes_AK);
     re_trans.last := re_trans.last + 1;
     retrans_ptr := re_trans.buf[re_trans.last];
     retrans_ptr.data := mes_AK;
     retrans_ptr.count := 0;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     set_window_timer;
     end;



trans
  provided (timer(data_ack))
    from OPEN to SAME            (* timer for acknowledge of
                                    receiving data *)

     begin
     r_ack_seq := rcv_seq;
     make_AK;
     output TP4M_O.TP4M_MESreq(mes_AK);
     end;



trans
  provided (timer(inact_ack))
    from OPEN to SAME            (* no DT/ED sent in the interval;
                                    sending AK to keep active *)

     begin
     make_AK;
     output TP4M_O.TP4M_MESreq(mes_AK);
     set_inact_ack_timer;
     end;
```

```
trans
  provided (timer(inact))
     from OPEN to CLOSING

     var
          retrans_ptr: retrans_type;

     begin
     stop_window_timer;
     stop_retrans_timer;
     make_DR(R_inact_timer);                (* not specified ! *)
     output TP4M_O.TP4M_MESreq(mes_DR);
     re_trans.total := 1;                   (* remove all retrans *)
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := mes_DR;
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     make_DRg(R_inact_timer);
     output TP4I_O.TP4I_MESreq(tp_mes);  (* send TDISind to linker *)
     end;



trans
  provided (timer(ref))
     from REFWAIT to CLOSED

     begin
     flag_kill_me := now
     end;
```

```
(***   GROUP-3:  TPDU from TRANSPORT GATEWAY   ***)


trans
  when  TP4I_I.TP4I_MES.ind                    (* CR from gateway *)
    provided (TP_code(message) = CRCODE)
      from CLOSED to WFCCiso

     var
          retrans_ptr: retrans_type;

     begin
     accept_TP_mes (message);              (* take in the TP_mes *)
     rcv_cdt := set_credit;
     trans_CRg;
     output TP4M_O.TP4M_MESreq(mes_CR);
     re_trans.total := 1;
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := mes_CR;
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     end;



trans
  when TP4I_I.TP4I_MESind
    provided (TP_code(message) = DRCODE)
      from WFCCiso

     var
          retrans_ptr: retrans_type;

     begin
     accept_TP_mes (message);               (* take in the TP_mes *)
     if (local_choice) then
          next_state := WBCL;
      else                                  (* not local choice *)
          begin
          stop_retrans_timer;
          accept_TP_mes (message);              (* take in the TP_mes *)
          trans_DRg;
          output TP4M_O.TP4M_MESreq(mes_DR);
          re_trans.total := 1;
          retrans_ptr := re_trans.buf[1];
          retrans_ptr.data := mes_DR;
          retrans_ptr.count := 1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          end;
     end;
```

```
trans
  when TP4I_I.TP4I_MESind
    provided (TP_code(message) = CCCODE)
     from WFCCout to AKWAITiso

     var
          retrans_ptr: retrans_type;

     begin
     accept_TP_mes (message);          (* take in the TP_mes *)
     rcv_cdt := set_credit;
     trans_CCg;
     output TP4M_O.TP4M_MESreq(mes_CC);
     re_trans.total := 1;
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := mes_CC;
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     end;



trans
  when TP4I_I.TP4I_MESind
    provided (TP_code(message) = DRCODE)
      from WFCCout to CLOSED

     begin
     accept_TP_mes (message);          (* take in the TP_mes *)
     trans_DRg;
     output TP4M_O.TP4M_MESreq(mes_DR);
     flag_kill_me := now;
     end;



trans
  when TP4I_I.TP4I_MESind
    provided (TP_code(message) = DRCODE)
      from AKWAITiso to CLOSING

     var
          retrans_ptr: retrans_type;

     begin
     stop_retrans_timer;
     accept_TP_mes (message);          (* take in the TP_mes *)
     trans_DRg;
     output TP4M_O.TP4M_MESreq(mes_DR);
```

```
      retrans_ptr := re_trans.buf[1];
      retrans_ptr.data := mes_DR;
      retrans_ptr.count := retrans_ptr.count+1;
      retrans_ptr.timer := RETRANS_TIME;
      set_retrans_timer;
      end;



trans
  when TP4I_I.TP4I_MESind
    provided (TP_code(message) = DTCODE)
      from AKWAIT to OPEN

      begin
      stop_inact_ack_timer;
      accept_TP_mes (message);          (* take in the TP_mes *)
      trans_DTg;
      output TP4M_O.TP4M_MESreq(mes_DT); (* send DT to TP4 Net *)
      retrans_ptr := re_trans.buf[1];
      retrans_ptr.data := mes_DT;
      retrans_ptr.count := retrans_ptr.count+1;
      retrans_ptr.timer := RETRANS_TIME;
      set_retrans_timer;
      set_window_timer;
      set_inact_timer;
      set_inact_ack_timer;
      end;



trans
  when TP4I_I.TP4I_MESind
    provided (TP_code(message) = EDCODE)
      from AKWAIT to OPEN

      begin
      stop_inact_ack_timer;
      exp_flag := true;                  (* to block future DTs *)
      accept_TP_mes (message);           (* take in the TP_mes *)
      trans_EDg;
      output TP4M_O.TP4M_MESreq(mes_ED); (* send ED to TP4 Net *)
      retrans_ptr := re_trans.buf[1];
      retrans_ptr.data := mes_ED;
      retrans_ptr.count := retrans_ptr.count+1;
      retrans_ptr.timer := RETRANS_TIME;
      set_retrans_timer;
      set_window_timer;
      set_inact_timer;
      set_inact_ack_timer;
      end;
```

```
trans
  when TP4I_I.TP4I_MESind
    provided (TP_code(message) = DRCODE)
      from OPEN to CLOSING

     var
          retrans_ptr: retrans_type;

     begin
     stop_window_timer;
     stop_inact_timer;
     stop_retrans_timer;
     accept_TP_mes (message);           (* take in the TP_mes *)
     trans_DRg;
     output TP4M_O.TP4M_MESreq(mes_DR);
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := mes_DR;
     retrans_ptr.count := retrans_ptr.count+1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     end;



trans
  when TP4I_I.TP4I_MESind
    provided (TP_code(message) = DTCODE)
     from OPEN to SAME

     begin
     stop_window_timer;
     stop_inact_timer;
     if (exp_flag=flase) then
          begin
          accept_TP_mes (message);           (* take in the TP_mes *)
          trans_DTg;
          output TP4M_O.TP4M_MESreq(mes_DT);
          retrans_ptr := re_trans.buf[re_trans.last];
          retrans_ptr.data := mes_DT;
          retrans_ptr.count := retrans_ptr.count+1;
          retrans_ptr.timer := RETRANS_TIME;
          re_trans.last := re_trans.last + 1;
          set_retrans_timer;
          end;
      else                                  (* blocked due to ED *)
          begin
          exp_que.last := exp_que.last + 1;
          exp_que.buf[exp_que.last] := message;
          end;
     set_window_timer;
     set_inact_timer;
     end;
```

```
trans
  when TP4I_I.TP4I_MESind
    provided (TP_code(message) = EDCODE)
     from OPEN to SAME

     begin
     stop_window_timer;
     stop_inact_timer;
     exp_flag := true;                  (* to block future DTs *)
     accept_TP_mes (message);          (* take in the TP_mes *)
     trans_EDg;
     output TP4M_O.TP4M_MESreq(mes_ED);
     retrans_ptr := re_trans.buf[re_trans.last];
     retrans_ptr.data := mes_ED;
     retrans_ptr.count := retrans_ptr.count+1;
     retrans_ptr.timer := RETRANS_TIME;
     re_trans.last := re_trans.last + 1;
     set_retrans_timer;
     set_window_timer;
     set_inact_timer;
     end;
```

APPENDIX A.3.2 DDN_TCP SUBMODULE

```
(***                                                    ***)
(***        DDN TCP Submodule Specification             ***)
(***                                                    ***)

body DDN_TCP_entity_body for DDN_TCP_entity_type;

  type

     TCP_HDR_type = record
          src_port   : short_word;        (* source port *)
          dst_port   : short_word;        (* destin port *)
          seq_num    : word_type;         (* sequence num *)
          ack_num    : word_type;         (* ack num *)
          offset     : octet;             (* header length *)
          flags      : octet;             (* TCP code *)
          window     : short_word;        (* window size *)
          checksum   : short_word;
          urg_ptr    : short_word;        (* urgent pointer *)
          options    : data_type;
          data       : data_type;         (* user data *)
          end;


     retrans_type = record
          timer : integer;     (* time left before retrans *)
          count : integer;     (* count of retrans *)
          data  : data_type;   (* PDU *)
          end;

     reorder_type = record
          seq  : integer;      (* sequence number for re-order *)
          data : data_type;    (* PDU *)
          end;
```

```
(***                                                    ***)
(***        TCP Interface Interaction Point             ***)
(***                                                    ***)


(* TP-4 State Machine Interaction Point *)

channel TCP_machine_primitives (user, provider);

  by privider:
     TCPM_MESind (message : IP_TP_message_type);
                                (* send TPDUs to TCP_machine *)

  by user:
     TCPM_MESreq (message : IP_TP_message_type);
                                (* TCP_machine sends TPDUs out *)


(* TP Transport Gateway Interaction Point (to linker module) *)

channel TCP_interface_primitives (user, provider);

  by provider:
     TCPI_MESind (message : IP_TP_message_type);

  by user:
     TCPI_MESreq (message : IP_TP_message_type);


(***                                                    ***)
(***    TCP State Machine SubModule Specification       ***)
(***                                                    ***)


module TCP_machine_type process
          (TCPM_I: TCP_machine_primitives (user);
           TCPM_O: TCP_machine_primitives (provider);
           TCPI_I: TCP_interface_primitives (user);
           TCPI_O: TCP_interface_primitives (provider);
          );
     export
          state      : TCP_state_type; (* connection state *)
          l_module   : octet;        (* local module id *)
          g_module   : octet;        (* GW module id *)
          d_ip_addr  : word_type;    (* destination IP address *)
          s_ip_addr  : word_type;    (* source IP address *)
          l_ref      : short_word;   (* local reference *)
          r_ref      : short_word;   (* remote reference *)
          g_ref      : short_word;   (* gateway cross reference *)
          end;


body TCP_machine_body for TCP_machine_type;  external;
```

```
(***                                         ***)
(***    variables of the DDN_TCP_body         ***)
(***                                         ***)


var

     total_conn : integer;    (* total active transport conn *)
     conn_flag  : array [1..MAX_CONN] of boolean;
                              (* slot available flag *)
     TCPM       : array [0..MAX_CONN] of TCP_machine_type;
                              (* instance of TCP_machine *)

     cur_conn: integer;       (* cur connection reference *)
     direction : direction_type; (* cur mes flow direction *)
     cur_mod_id : octet;      (* module Id *)
     cur_mes_id : octet;      (* message Id *)
     cur_mes : IP_TP_message_type; (* cur IP_TP message *)
     cur_tpdu : TPDU_message_type;    (* TPDU, used as pointer *)
     cur_tcp : TCP_HDR_type;  (* TCP hdr, used as pointer *)
     cur_s_IP : word_type;    (* source IP address *)
     cur_d_IP : word_type;    (* destin IP address *)
     cur_tp_code : TPDU_code_type; (* current TPDU code *)
     src_port,dst_port: short_word;

     total_mes : integer;     (* total messages going through *)
     err_mes : integer;       (* messages with error  conditions *)
     fatal_err : integer;     (* fatal errors *)
```

```
(***                                            ***)
(***  Common Procedures of DDN_TCP_body         ***)
(***                                            ***)


function conn_alloc : integer;

     (* allocate a free slot of connection *)

  var
     i,j : integer;

     begin
     j := 0;
     if (total_conn < MAX_CONN) then
       for i := MAX_CONN to 1 do
         if (conn_flag[i] = false) then
           j := i;
     if (j > 0) then
          begin
          total_conn := total_conn+1;
          conn_flag[j] := true;
          end;
     conn_alloc := j;
     end;

(* In the case of (total_conn=MAX_CONN), (conn_alloc=0) will be
returned.  There exists a TCPM[0] to be used in this case to
respond to abnormal TPDUs. *)


procedure conn_free (j : integer);  (* free the connection slot *)

     begin
     if ((j>0) and (j<=MAX_CONN)) then
       if (conn_flag[j] = true) then
          begin
          conn_flag[j] := flase;
          total_conn := total_conn-1;
          end;
     end;


function TP_code (TPDU: TPDU_type): TPDU_code_type;

     (* check the TPDU_code of the TPDU *)

     begin
     TP_code := TPDU[2];               (* 4 bits at left *)
     end;
```

```
function cur_TP_code (mes: IP_TP_message) : TPDU_code_type;

     (* check the TPDU_code of the 1st IP_TP message in queue *)

     begin
     cur_mes := mes;                              (* access NQ message *)
     cur_tpdu := cur_mes.data;                    (* point to TPDU *)
     cur_conn := cur_tpdu.dst_ref;                (* TCP reference index *)
     cur_tp_code := TP_code(cur_tpdu);            (* get the TPDU code *)
     cur_TP_code := cur_tp_code;
     end;


function TCP_code (byte: octet): TPDU_code_type;

     (* check the TPDU_code of the TCP hdr *)

     begin
     case byte of
       0: TCP_code := DTCODE;                     (*  *)
       1: TCP_code := DRCODE;                     (* FIN *)
       2: TCP_code := CRCODE;                     (* SYN *)
       4: TCP_code := RJCODE;                     (* RST *)
       8: TCP_code := DTCODE;                     (* PSH and data *)
       16: TCP_code := AKCODE;                    (* ACK *)
       18: TCP_code := CCCODE;                    (* SYN,ACK *)
       32: TCP_code := DTCODE;                    (* UGR and data *)
       40: TCP_code := DTCODE;                    (* URG,PSH, and data *)
       56: TCP_code := AKCODE;                    (* URG,PSH,ACK *)
       end;
     end;


function cur_TCP_code (mes: IP_TP_message) : TPDU_code_type;

     (* check the TCP code of the 1st IP_TP message in queue *)

     begin
     cur_mes := mes;
     cur_tcp := mes.data;                          (* point to TCP hdr *)
     cur_tp_code := TCP_code(cur_tcp.flags); (* get the TPDU code *)
     if ((cur_tp_code=AKCODE) and d_length(cur_tcp.data)>0)) then
          cur_tp_code = DTCODE;
     cur_TCP_code := cur_tp_code;
     end;
```

```
function tcp_acceptable (mes: IP_TP_message_type) : boolean;

     (* check the TCP header to see if it is acceptable *)

  var
     ok: boolean;
     i,j,k,l: integer;

     begin
     ok := chk_checksum(mes.data, mes.data.offset*4);
     if (ok) then
          begin
          cur_TCP_code(mes);            (* get cur_tp_code, etc. *)

          cur_conn := 0;
          i := 0;
          while ((i<total_conn) and (cur_conn=0)) do
               begin
               if((con_flag[i]=true) and
                  (TCPM[i].d_ip_addr=mes.d_IP_addr) and
                  (TCPM[i].s_ip_addr=mes.s_IP_addr) and
                  (TCPM[i].l_port=mes.data.dst_port) and
                  (TCPM[i].r_port=mes.data.src_port)) then
                    cur_conn := i;
                else
                    i := i + 1;
               end;

          if(cur_conn>0) then
               begin
               if (TCPM[cur_conn].g_module=0) then
                    TCPM[cur_conn].g_module = cur_mes.module/16;
                else
                  if(TCPM[cur_conn].g_module<>cur_mes.module/16) then
                    ok := flase;
               if(TCPM[cur_conn].r_ref=0) then
                    TCPM[cur_conn].r_ref := cur_tpdu.src_ref;
                  else
                   if (cur_tpdu.src_ref<>TCPM[cur_conn].r_ref) then
                    ok := false;              (* check the SRC_REF *)
               end;
          end;

     tcp_acceptable := ok;
     end;
```

```
function dup_CR (mes: IP_TP_message_type) : boolean;

     (* check the CR TPDU to see if it is duplicate *)

     var
          dup: boolean;
     begin
     cur_conn := 0;
     dup := false;
     i := 1;
     while ((i<=MAX_CONN) and (dup=false)) do
          begin
          if ((conn_flag[i]=true) and
              (cur_mes.s_IP_addr=TCPM[i].s_ip_addr) and
              (cur_mes.d_IP_addr=TCPM[i].d_ip_addr) and
              (TCPM[i].l_port=cur_mes.data.dst_port) and
              (TCPM[i].r_port=cur_mes.data.src_port)) then
                    begin
                    cur_conn := i;
                    dup := true;
                    end;
          i := i+1;
          end;
     dup_CR := dup;
     end;


function dup_CRg (mes: TPDU_message_type) : boolean;

     (* check the CR TPDU from TP_GW to see if it is duplicate *)

     var
          dup: boolean;
          i: integer;
     begin
     cur_conn := 0;
     dup := false;
     i := 1;
     while((i<=MAX_CONN) and (dup=false)) do
          begin
          if ((conn_flag[i]=true) and
              (mes.d_IP_addr=TCPM[i].s_ip_addr) and
              (mes.s_IP_addr=TCPM[i].d_ip_addr) and
              (TCPM[i].l_port=mes.data.dst_port) and
              (TCPM[i].r_port=mes.data.src_port) and
              (mes.src_ref=TCPM[i].g_ref)) then
                    begin
                    cur_conn := i;
                    dup := true;
                    end;
          i := i+1;
          end;
     dup_CRg := dup;
     end;
```

```
(***                                                   ***)
(***     Initialization of DDN_TCP_body                ***)
(***                                                   ***)

initialize

     total_mes := 0;             (* total messages *)
     err_mes := 0;               (* error messages *)
     fatal_err := 0;             (* fatal errors *)

     total_conn := 0;            (* total transport connections *)
     for i := 1 to MAX_CONN do
          conn_flag := false; (* the connection slot is free *)

     init TCPM[0] with TCP_machine_interface_body();
                                 (* virtual State_machine for
                                    responding to error CR's *)
     TCPM[0].l_ref := 0;
     connect TCPM_I to TCPM[0].TCPM_I;
     connect TCPI_I to TCPM[0].TCPI_I;
     attach TCPM_O to TCPM[0].TCPM_O;
     attach TCPI_O to TCPM[0].TCPI_O;

     end;
```

```
(***                                          ***)
(***           State Transitions              ***)
(***                                          ***)

trans
  when NSi.IP_TP_MESind priority 0      (* receive from DDN-IP *)

     begin
     total_mes := total_mes+1;
     direction := in;

     if (tcp_acceptable(message)=false) then
          err_mes := err_mes+1;           (* TPDU not acceptable *)
       else if (cur_tp_code=CRCODE) then  (* and cur_conn=0 *)
          begin                           (* Connection Request *)
          if (dup_CR(message)=false) then
               begin
               cur_conn := conn_alloc();
               if (cur_conn>0) then
                    begin             (* initialize TCP protocol
                                         machine *)
                    init TCPM[cur_conn] with
                                   TCP_machine_interface_body();
                    with TCPM[cur_conn] do
                         begin
                         l_ref := cur_conn;
                         flag_kill_me := conn_in_progress;
                         end;
                    connect TCPM_I to TCPM[cur_conn].TCPM_I;
                    connect TCPI_I to TCPM[cur_conn].TCPI_I;
                    attach TCPM_O to TCPM[cur_conn].TCPM_O;
                    attach TCPI_O to TCPM[cur_conn].TCPI_O;
                    end;  (* if conn *)
               end; (* if dup_CR *)
          output TCPM[cur_conn].TCPM_I(message);
                (* pass TPDU for further check or response *)
          end;  (* if CR *)
       else
          output TCPM[cur_conn].TCPM_I(message);
     end;  (* trans *)

  (* For duplicated CR, the state machine TCPM[cur_conn] will
     response to it with CC.

     In case there is no local connection slot available,
     cur_conn is reset to 0 by dup_CR(), or by conn_alloc(), the
     CR will be passed to TCPM[0] which will in turn send a DR to
     refuse the connection request. *)
```

```
trans
  when TGS.TP_MESreq priority 0    (* receive from TCP, begore
                                      feeding into  TCP_machine *)

     begin
     total_mes := total_mes+1;
     direction := out;
     cur_mes := message;
     cur_tpdu := cur_mes.data;                (* point to TPDU *)
     cur_conn := cur_tpdu.src_ref;            (* TCP reference index *)
     cur_tp_code := TP_code(cur_tpdu);        (* get the TPDU code *)
     case cur_tp_code of
       CCCODE,DRCODE,DCCODE,DTCODE,
       AKCODE,ERCODE,RJCODE :
          output TCPM[cur_conn].TCPI_I(message);

       CRCODE :
          begin
          if (dup_CRg(message)=false) then
               begin
               cur_conn := conn_alloc();
               if (cur_conn>0) then
                    begin    (* initialize a new TCP_machine *)
                    init TCPM[cur_conn] with
                                 TCP_machine_interface_body();
                    with TCPM[cur_conn] do
                         begin
                         l_ref := cur_conn;
                         flag_kill_me := conn_in_progress;
                         end;
                    connect TCPM_I to TCPM[cur_conn].TCPM_I;
                    connect TCPI_I to TCPM[cur_conn].TCPI_I;
                    attach TCPM_O to TCPM[cur_conn].TCPM_O;
                    attach TCPI_O to TCPM[cur_conn].TCPI_O;
                    end;  (* if conn *)
               end; (* if *)
          output TCPM[cur_conn].TCPI_I(message);
                          (* pass TPDU for further checking *)
          end;  (* case CR *)
       end;  (* case *)
     end;  (* trans *)
```

```
trans
  any TCPM[i]: TCP_machine_interface_type do
    provided (TCPM[i].flag_kill_me = now) priority 1

     begin
     disconnect TCPM[i].TCPM_I;  (* disconnect port relations *)
     disconnect TCPM[i].TCPI_I;
     disattach TCPM[i].TCPM_O;
     disattach TCPM[i].TCPI_O;
     release TCPM[i];            (* terminate TCP_machine *)
     conn_free(i);               (* free the slot for future use *)
     end;

end;  (* of DDN_TCP_body *)
```

```
(***                                                         ***)
(***    TCP_machine_interface_body  Specification            ***)
(***                                                         ***)

body TCP_machine_interface_body for TCP_machine_interface_type;

type

     TCP_state_type = (CLOSED, ESTAB, SYN_SENTddn,
                       SYN_SENTout, SYN_RCVDddn, SYN_RCVDout,
                       FIN_WAITddn, FIN_WAITout, TIME_WAIT);

     OPTION_END = 0;
     OPTION_NO = 1;
     OPTION_SIZE = 2;


var
     proto_err : integer;     (* fatal errors *)
     l_port    : short_word;  (* local port id *)
     r_port    : short_word;  (* remote port id *)
     initiator : initiator_type; (* who initiate the conn *)
     size      : octet;       (* max TPDU size *)
     g_size    : octet;       (* GW max TPDU size *)
     snd_nxt  : word_type;   (* sending sequence no *)
     snd_wnd  : short_word;  (* sending credit *)
     snd_ack : word_type;   (* ack_ed sequence no *)
     rcv_nxt   : word_type;   (* receiving sequence no *)
     rcv_wnd   : short_word;  (* receiving credit *)
     rcv_ack : word_type;   (* ack_ed sequence no *)

     re_trans   : queue_type of retrans_type; (* retransmission *)
     re_order   : queue_type of reorder_type; (* re-order *)

     cur_mes : IP_TP_message_type; (* cur IP_TP message *)
     cur_tpdu : TPDU_type;         (* TPDU, used as pointer *)
     cur_tcp:  TCP_HDR_type;       (* TCP hdr *)
     cur_tp_code : TPDU_code_type; (* current TPDU code *)
     var_ptr: ^TP_var_part_type;   (* ptr to variable part *)
     sub_state      : boolean;     (* associated flag *)
     seq,ack : word_type;
     wnd : short_word;


     tp_mes : TPDU_message_type;
```

```
(***                                          ***)
(***          Common Procedures               ***)
(***                                          ***)


procedure set_timer (FSM: TCP_machine_type;
                     time: integer;
                     kind: timer_type);      primitive;

     (* set timer *)


procedure set_retrans_timer;

     (* set re-trans timer *)

     begin
     set_timer (TCPM[l_ref], RETRANS_TIME, retrans);
     end;


procedure set_ref_timer;

     (* set ref timer *)

     begin
     set_timer (TCPM[l_ref], REF_TIME, ref);
     end;


procedure set_window_timer;

     (* set window timer *)

     begin
     set_timer (TCPM[l_ref], WINDOW_TIME, window);
     end;


procedure set_inact_timer;

     (* set inact timer *)

     begin
     set_timer (TCPM[l_ref], INACT_TIME, inact);
     end;
```

```
procedure stop_timer (FSM: TCP_machine_type;
                      kind: timer_type);      primitive;

     (* stop timer *)


procedure stop_retrans_timer;

     (* stop retrans timer *)

     begin
     stop_timer (TCPM[l_ref], retrans);
     end;


procedure stop_window_timer;

     (* stop window timer *)

     begin
     stop_timer (TCPM[l_ref], window);
     end;


procedure stop_inact_timer;

     (* stop inact timer *)

     begin
     stop_timer (TCPM[l_ref], inact);
     end;


function timer (kind: timer_type): logical;      primitive;

     (* dst at the timer interrupt *)


function set_window: integer;  primitive;

     (* set window size for local TCPM receiving.  It is used to
          get the window width for the flow control.   The
          function can be implemented by estimating the available
          storage space in memory, dividing them between the
          modules in the system, and between the active sessions
          in the module. *)
```

```
function trans_d_port(addr: short_word) : short_word; primitive;

          (* translate DDN_TCP port address to GW_TP address *)


function trans_g_port(addr: short_word) : short_word; primitive;

          (* translate GW_TP address to DDN_TCP port address *)


function cur_TCP_code (mes: IP_TP_message_type) : TPDU_code_type;

     (* check the TPDU_code of message from IP *)

     begin
     cur_mes := mes;
     cur_tcp := cur_mes.data;
     cur_tp_code := TCP_code(cur_tcp.flags);      (* 4 bits at left *)
     cur_TCP_code := cur_tp_code;
     end;


function TP_code (mes: TPDU_message_type) : TPDU_code_type;

     (* check the TPDU_code of message from TP-GW *)

     begin
     tp_mes := mes;
     cur_tp_code := tp_mes.tpdu_code;    (* 4 bits at right *)
     TP_code := cur_tp_code;
     end;


procedure tcp_checksum (var mes: IP_TP_message_type);

     (* complete the TPDU with checksum, etc. *)

     begin
     mes.data := set_checksum(cur_tcp, cur_tcp.offset*4, 17);
     end;
```

```
procedure accept_TPDU (mes: IP_TP_message_type);

     (* accept the control info in the TPDU from DDN-TCP *)

     var
          sn: word_type;

     begin
     cur_mes := mes;
     cur_tpdu := cur_mes.data;
     cur_tp_code := cur_tpdu[2] / 16;       (* 4 bits at left *)

     sub_state := true;                    (* true for OK *)
     if (s_ip_addr=0) then
          s_ip_addr  := cur_mes.s_IP_addr;  (* fill in source IP *)
       else if (s_ip_addr<>cur_mes.s_IP_addr) then
          sub_state := false;

     if (d_ip_addr=0) then
          d_ip_addr  := cur_mes.d_IP_addr;  (* fill in destin IP *)
       else if (d_ip_addr<>cur_mes.d_IP_addr) then
          sub_state := false;

     if (l_port=0) then
          l_port := cur_tpdu.dst_port;
      else if (l_port<>cur_tpdu.dst_port) then
          sub_state := false;

     if (r_port=0) then
          r_port := cur_tpdu.src_port;
      else if (r_port<>cur_tpdu.src_port) then
          sub_state := false;

     seq := cur_tpdu.seq;
     ack := cur_tpdu.ack;
     wnd := cur_tpdu.wnd;

     sn := 0;
     i := 1;
     var_ptr := cur_tpdu.options;
     l := cur_tpdu.offset*4-20;            (* length of options *)
     while (l>0) do
          begin
          case var_ptr[l] of
            OPTION_END :       (* 0000 0000: No more *)
               l := 0;
            OPTION_SIZE:       (* 0000 0002: Max PDU size *)
               begin
               sn := get_word (var_ptr, 3);
               l := l-3;
               i := i+3;
               end;
          end;
            default :                  (* CHKSUMCODE,REASSIGN,etc *)
```

```
                  null;
               end; (* case *)

            var_ptr := ^var_ptr[2];
            l := l-1;
            end; (* while *)
       end;



procedure accept_TP_mes(mes: TPDU_message_type);

       (* accept the control info in the TPDU from TCP *)

   var
       n: word_type;
       sn: short_word;
       i,j,k,l: integer;
       var_ptr: ^TP_var_part_type;

       begin
       tp_mes := mes;
       cur_tp_code := tp_mes.tpdu_code; (* 4 bits at right *)
       if (g_ref=0) then
            g_ref := tp_mes.src_ref;       (* fill in remote ref *)
       sub_state := true;                  (* true for OK *)

       if (cur_tp_code=CRCODE) then
            begin
            g_module := tp_mes.module mod 16;
            d_ip_addr := tp_mes.d_IP_addr;
            s_ip_addr := tp_mes.s_IP_addr;
            end;

       var_ptr := ^tp_mes.var_part;
       l := v_length;                      (* length of var. part *)
       while (l>0) do
            begin
            k := var_ptr.length;
            case var_ptr.param_code of

              CALINGCODE :      (* 1100 0001: src port-ID *)
                   if (cur_tp_code=CRCODE) then
                      l_port := get_sword (var_ptr.value, k);
                   else if (cur_tp_code<>CCCODE) then
                      sub_state := false;
                   else if (l_port=0) then
                      l_port:=get_sword(var_ptr.value,k);
                   else if (l_port<>get_sword(var_ptr.value,k))) then
                      sub_state := false;

              CALLEDCODE :      (* 1100 0010: dst port-ID *)
                   if (cur_tp_code=CRCODE) then
                      r_port := get_sword (var_ptr.value, k);
```

```
            else if (cur_tp_code<>CCCODE) then
               sub_state := false;
            else if (r_port=0) then
               r_port:=get_sword(var_ptr.value,k);
            else if (r_port<>get_sword(var_ptr.value,k))) then
               sub_state := false;

       PDUSIZCODE :        (* 1100 0000: TPDU size *)
            if (cur_tp_code=CRCODE) then
               g_size := get_sword (var_ptr.value, k);
            else if (cur_tp_code<>CCCODE) then
               sub_state := false;
            else
               begin
               sn := get_sword(var_ptr.value,k);
               if ((g_size=0) or (g_size>sn)) then
                    g_size := sn;
               end;

     var_ptr := ^var_ptr.value[k+1];
     l := l-k-2;
     end;
end;
```

```
procedure build_TCP;

     (* filling in the parts of TCP header, and GW message header *)

     begin
     cur_tcp.src_port := l_port;
     cur_tcp.dst_port := r_port;
     cur_tcp.seq_numt := snd_nxt;
     cur_tcp.ack_numt := rcv_nxt;
     cur_tcp.window := rcv_wnd;
     tcp_checksum(cur_tcp);

     cur_mes.module := l_module;
     cur_mes.s_IP_addr := d_ip_addr;
     cur_mes.d_IP_addr := s_ip_addr;
     cur_mes.data := cur_tcp;
     end;


procedure build_ER;

     (* build ER for DDN-TCP, in cases of any protocol error *)

     begin
     cur_tcp.flags := RST;
     cur_tcp.offset := 5;
     build_TCP;
     end;


procedure build_DC;

     (* build DC for DDN-TCP in response to:
          FIN-WAIT * FIN *)

     begin
     cur_tcp.flags := FIN+ACK;
     cur_tcp.offset := 5;
     build_TCP;
     end;
```

```
procedure build_CC;

     (* build CC for DDN-TCP *)

     begin
     cur_tcp.flags := SYN+ACK;
     cur_tcp.offset := 5;
     build_TCP;
     end;



procedure build_AK;

     (* make up AK for DDN-TCP, in cases of:
          ESTAB * DT,
          ESTAB * timers *)

     begin
     update_window();          (* update window control info *)
     cur_tcp.flags := cur_tcp.flags + ACK;
     cur_tcp.offset := 5;
     build_TCP;
     end;


function opt_calling(port: short_word): data_type;

     (* make up calling TASP option *)

     var
          ptr: data_type;

     begin
     ptr := d_create(4);
     ptr[1] := CALINGCODE;
     ptr[2] := 2;
     d_puts(ptr,3,d_encode2(port));
     opt_calling := ptr;
     end;
```

```
function opt_called(port: short_word): data_type;

     (* make up called TASP option *)

     var
          ptr: data_type;

     begin
     ptr := d_create(4);
     ptr[1] := CALLEDCODE;
     ptr[2] := 2;
     d_puts(ptr,3,d_encode2(port));
     opt_called := ptr;
     end;


procedure trans_TCP;

     (* common operation to translate TCP TPDUs into TP GW format *)

     begin
     tp_mes.module := cur_mes.module;
     tp_mes.message := cur_mes.message;
     tp_mes.s_IP_addr := cur_mes.s_IP_addr;
     tp_mes.d_IP_addr := cur_mes.d_IP_addr;

     tp_mes.dst_ref := g_ref;
     tp_mes.src_ref := l_ref;
     tp_mes.class := 4;

     tp_mes.v_length := d_length(tp_mes.var_part);
     tp_mes.d_length := d_length(cur_tcp.data);
     tp_mes.data := d_create(tp_mes.d_length);
     d_puts (tp_mes.data, 1, cur_tcp.data);
     end;


procedure trans_CR;

     (* translate CR of DDN-TCP to GW format, in case of
          CLOSED * CR  *)

     begin
     tp_mes.code := CRCODE;
     tp_mes.var_part := opt_calling(trans_d_port(l_port));
     tp_mes.var_part := d_append(opt_called(trans_d_port(r_port)));
     trans_TCP;
     end;
```

```
procedure trans_CC;

     (* translate CC of DDN-TCP to GW format, in case of
          SYN_SENTddn * CC  *)

     begin
     tp_mes.code := CCCODE;
     tp_mes.var_part := opt_calling(trans_d_port(l_port));
     tp_mes.var_part := d_append(opt_called(trans_d_port(r_port)));
     trans_TCP;
     end;


procedure trans_DR;

     (* translate DR of DDN-TCP to GW format, in case of
          ESTAB * FIN *)

     begin
     tp_mes.code := DRCODE;
     tp_mes.var_part := opt_calling(trans_d_port(l_port));
     tp_mes.var_part := d_append(opt_called(trans_d_port(r_port)));
     trans_TCP;
     end;


procedure trans_DT;

     (* translate DT of DDN-TCP to GW format, in case of
          SYN_RCVDddn, ESTAB * DT  *)

     begin
     tp_mes.code := DTCODE;
     tp_mes.var_part := opt_calling(trans_d_port(l_port));
     tp_mes.var_part := d_append(opt_called(trans_d_port(r_port)));
     trans_TCP;
     end;


procedure trans_CRg;

     (* translate CR of GW format to DDN-TCP format, in case of
          CLOSED * CRg  *)

     begin
     update_window();          (* update window control info *)
     cur_tcp.flags := SYN;
     cur_tcp.offset := 5;
     build_TCP;
     end;
```

```
procedure trans_CCg;

     (* translate CC of GW format to DDN-TCP format, in case of
          SYN_SENTout * CCg  *)

     begin
     update_window();         (* update window control info *)
     cur_tcp.flags := SYN + ACK;
     cur_tcp.offset := 5;
     build_TCP;
     end;


procedure trans_DRg;

     (* translate DR of GW format to DDN-TCP format, in case of
          SYN_SENTddn, SYN_SENTout, SYN_RCVDddn, ESTAB * DRg  *)

     begin
     cur_tcp.flags := FIN
     cur_tcp.offset := 5;
     build_TCP;
     end;


procedure trans_DTg;

     (* translate DT of GW format to DDN-TCP format, in case of
          ESTAB * DTg  *)

     begin
     cur_tcp.flags := 0;
     cur_tcp.offset := 5;
     build_TCP;
     end;


procedure check_queue;   primitive;

     (* Check all the TPDUs in the re-trans queue for re-
          transmission if necessary. *)
```

```
(***                                        ***)
(***            Initialization              ***)
(***                                        ***)


initialize

    state := CLOSED;
    proto_err := 0;




(***                                        ***)
(***     Transitions of TCP_machine         ***)
(***                                        ***)

(***   GROUP-1:  TPDU from DDN-TCP     ***)

(***   State_transition from  CLOSED   ***)


trans
  when TCPM_I.TCPM_MESind                       (* CR from TCP *)
    provided ((cur_TCP_code(message)=CRCODE)) and (l_ref=0))
      from CLOSED to same

    begin                                   (* no conn available *)
    build_ER;
    output TCPM_O.TCPM_MESreq(cur_mes);
    end;



trans
  when TCPM_I.TCPM_MESind                       (* CR from TCP *)
    provided ((cur_TCP_code(message)=CRCODE)) and (l_ref<>0))
      from CLOSED

    begin
    accept_TPDU (message);          (* take in the TPDU *)
    snd_nxt := time_of_day();
    snd_wnd := 0;
    snd_ack := 0;

    rcv_nxt := seq;
    rcv_wnd := wnd;
    rcv_ack := 0;
    trans_CR;
    output TCPI_O.TCPI_MESreq(tp_mes); (* send translated
                                          message to linker *)
    next_state := SYN_SENTout;
    end;
```

```
trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = CCCODE)
      from CLOSED to SAME

     begin
     build_ER;
     output TCPM_O.TCPM_MESreq(cur_mes);
     end;



trans
  when TCPM_I.TCPM_MESind
    provided ((cur_TCP_code(message) = DTCODE) or
              (cur_TCP_code(message) = AKCODE))
     from CLOSED to SAME

     begin
     proto_err := proto_err+1;
     build_ER;
     output TCPM_O.TCPM_MESreq(cur_mes);
     end;


(***   State_transition from  SYN_SENTddn   ***)


trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = CCCODE)
      from SYN_SENTddn

     var
          retrans_ptr := ^retrans_type;

     begin
     stop_retrans_timer;                 (* for CR *)
     accept_TPDU (message);              (* take in the TPDU *)
     rcv_nxt := seq;
     rcv_wnd := wnd;
     rcv_ack := 0;

     re_trans.total := re_trans.total-1;  (* remove CR from
                                                  retrans_queue *)
     set_window_timer;
     set_inact_timer;
     trans_CC;
     output TCPI_O.TCPI_MESreq(tp_mes); (* send translated
                                                 message to linker *)
     next_state := SYN_RCVDout;
     end;
```

```
trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = DRCODE)
      from SYN_SENTddn to TIME_WAIT

     begin
     accept_TPDU (message);          (* take in the TPDU *)
     set_ref_timer;
     trans_DR;
     output TCPI_O.TCPI_MESreq(tp_mes);  (* send TDISind to linker *)
     end;



trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = ERCODE)
      from SYN_SENTddn to TIME_WAIT

     begin
     accept_TPDU (message);            (* take in the TPDU *)
     set_ref_timer;
     build_DRg;
     output TCPI_O.TCPI_MESreq(tp_mes);  (* send TDISind to linker *)
     end;


(***   State_transition from  SYN_SENTout   ***)


trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = DRCODE)
     from SYN_SENTout to CLOSED

     begin
     accept_TPDU (message);               (* take in the TPDU *)
     build_DC;
     output TCPM_O.TCPM_MESreq(cur_mes);
     trans_DR;
     output TCPI_O.TCPI_MESreq(tp_mes); (* send TDISind to linker *)
     flag_kill_me := now;               (* terminate the TPM *)
     end;
```

```
(***   State_transition from  SYN_RCVD   ***)


trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = CRCODE)
      from SYN_RCVD to SAME

     var
          retrans_ptr: retrans_type;

     begin
     stop_retrans_timer;
     retrans_ptr := re_trans.buf[1];
     output TCPM_O.TCPM_MESreq(retrans_ptr.data);  (* retrans CC *)
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     end;



trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = ERCODE)
     from SYN_RCVD to FIN_WAIT

     var
          retrans_ptr: retrans_type;

     begin
     stop_retrans_timer;
     accept_TPDU (message);          (* take in the TPDU *)
     retrans_ptr := re_trans.buf[1];
     build_DR;
     output TCPM_O.TCPM_MESreq(cur_mes);
     retrans_ptr.data := cur_mes;         (* replace CC with DR *)
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     build_DRg;
     output TCPI_O.TCPI_MESreq(tp_mes);  (* send TDISind *)
     end;
```

```
trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = DRCODE)
      from SYN_RCVD to TIME_WAIT

     begin
     accept_TPDU (message);          (* take in the TPDU *)
     build_DC;
     output TCPM_O.TCPM_MESreq(cur_mes);
     set_ref_timer;
     trans_DR;
     output TCPI_O.TCPI_MESreq(tp_mes);  (* send TDISind *)
     end;


(***   State_transition from  SYN_RCVDddn   ***)


trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = DTCODE)
     from SYN_RCVDddn to ESTAB

     begin
     stop_retrans_timer;                 (* no more re-trans CC *)
     re_trans.last := 0;                 (* clear re-trans queue *)

     accept_TPDU (message);              (* take in the TPDU *)
     trans_DT;
     output TCPI_O.TCPI_MESreq(tp_mes);
     set_inact_timer;
     set_inact_ack_timer;            (* for sending AK at suitable
                                         interval in absence of DT *)
     end;



(***   State_transition from  ESTAB  ***)


trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = CRCODE)
      from ESTAB to SAME

     begin
     stop_inact_timer;
     set_inact_timer;
     end;
```

```
trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = CCCODE)
      from ESTAB to SAME

     begin                                          (* duplicated CC *)
     stop_inact_timer;
     build_AK;
     output TCPM_O.TCPM_MESreq (cur_mes);       (* repeat AK *)
     set_inact_timer;
     end;


trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = ERCODE)
     from ESTAB to SAME

     var
          retrans_ptr: retrans_type;

     begin
     stop_window_timer;
     stop_inact_timer;
     stop_retrans_timer;
     accept_TPDU (message);            (* take in the TPDU *)
     build_CR;
     output TCPM_O.TCPM_MESreq(cur_mes);       (* Re-synchronize *)
     end;


trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = DRCODE)
      from ESTAB to TIME_WAIT

     var
          retrans_ptr: retrans_type;

     begin
     stop_retrans_timer;
     accept_TPDU (message);            (* take in the TPDU *)
     build_DC;
     output TCPM_O.TCPM_MESreq(cur_mes);
     build_DR;                         (* TCP has order-release! *)
     output TCPM_O.TCPM_MESreq(cur_mes);
     set_ref_timer;
     end;
```

```
trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = DTCODE)
     from ESTAB to SAME

     begin
     stop_inact_timer;
     accept_TPDU (message);                  (* take in the TPDU *)

     if (seq = rcv_nxt) then                 (* DT in sequence *)
          begin
          trans_DT;
          output TCPI_O.TCPI_MESreq(tp_mes);
          set_data_ack_timer;                (* set timer to ack
                                                  during A_1 *)
          rcv_nxt := rcv_nxt + d_length(cur_tcp.data);
          while ((re_order.last>0) and    (* check re-seq queue *)
              (rcv_nxt=re_order.buf[1].seq)) do
               begin
               message := re_order.buf[1].data;
               trans_DT;
               output TCPI_O.TCPI_MESreq(tp_mes);
               rcv_nxt := rcv_nxt + d_length(re_order.buf[1].data);
               for i:=2 to re_order.last do
                    re_order.buf[i-1] := re_order.buf[i];
               re_order.last := re_order.last -1;
          end;
      else if((seq<rcv_nxt) or (seq>rcv_nxt+rcv_wnd)) then
                                                  (* out of window *)
          begin
          build_ER;
          output TCPM_O.TCPM_MESreq(cur_mes);
          end;
      else
          begin                         (* not-in-seq handling *)
          j := 0;
          for i := 1 to re_order.last do
            if ((seq>re_order.buf[i].seq) and (j=0)) then
               j := i;
          for i:=j to re_order.last do
               re_order.buf[i+1] := re_order.buf[i];
          re_order.buf[i].seq := seq;
          re_order.buf[i].data := message;   (* queued *)
          end;
     set_inact_timer;
     end;
```

```
trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = AKCODE)
     from ESTAB to SAME

     begin
     stop_inact_timer;
     stop_retrans_timer;
     accept_TPDU (message);                  (* take in the TPDU *)

     if (seq > snd_ack) then              (* AK in sequence *)
          begin
          snd_ack := seq;
          while ((re_trans.last>0) and    (* check re-seq queue *)
              (snd_ack>=re_trans.buf[1].seq)) do
               begin
               for i:=2 to re_trans.last do
                    re_trans.buf[i-1] := re_trans.buf[i];
               re_trans.last := re_trans.last -1;
          end;
                                (* else, automatic drop the AK *)
     if (re_trans.last>0) then
          set_retrans_timer;
     set_inact_timer;
     end;




(***   State_transition from  FIN_WAIT   ***)


trans
  when TCPM_I.TCPM_MESind
    provided ((cur_TCP_code(message) = CRCODE) or
              (cur_TCP_code(message) = CCCODE) or
              (cur_TCP_code(message) = DTCODE) or
              (cur_TCP_code(message) = AKCODE))
      from FIN_WAIT to SAME

  var
     reason: reason_type;

     begin
     accept_TPDU (message);              (* take in the TPDU *)
     build_ER;
     output TCPM_O.TCPM_MESreq(cur_mes);
     end;
```

```
trans
  when TCPM_I.TCPM_MESind
    provided ((cur_TCP_code(message) = DRCODE) or
              (cur_TCP_code(message) = DCCODE) or
              (cur_TCP_code(message) = ERCODE) )
      from FIN_WAIT to TIME_WAIT

     begin
     set_ref_timer;
     end;


(***   State_transition from  TIME_WAIT   ***)


trans
  when TCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = CCCODE)
      from TIME_WAIT to SAME

     begin
     accept_TPDU (message);          (* take in the TPDU *)
     build_ER;
     output TCPM_O.TCPM_MESreq(cur_mes);
     end;



trans
  whenTCPM_I.TCPM_MESind
    provided (cur_TCP_code(message) = DRCODE)
      from TIME_WAIT to SAME

     begin
     accept_TPDU (message);          (* take in the TPDU *)
     build_DC;
     output TCPM_O.TCPM_MESreq(cur_mes);
     end;
```

```
(***   GROUP-2:  timer-related   ***)

trans
  from SYN_SENTddn
    provided (timer(retrans))

     var
          retrans_ptr: ^retrans_type;

     begin                          (* only CR in retrans queue *)
     retrans_ptr := re_trans.buf[1];
     if (retrans_ptr.count < RETRANS_MAX) then
          begin
          output TCPM_O.TCPM_MESreq(cur_mes); (* re-trans CR *)
          retrans_ptr.count := retrans_ptr.count+1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          next_state := SAME;
          end;
      else begin                              (* count = max *)
          stop_window_timer;
          stop_inact_timer;
          build_DR;
          output TCPM_O.TCPM_MESreq(cur_mes);
          retrans_ptr.data := cur_mes;
          retrans_ptr.count := 1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          build_DRg;
          output TCPI_O.TCPI_MESreq(tp_mes); (* send TDISind *)
          next_state := FIN_WAIT;
          end;
     end;
```

```
trans
  provided (timer(retrans))
     from SYN_RCVD

     var
          retrans_ptr: retrans_type;

     begin
     retrans_ptr := re_trans.buf[1];
     if(retrans_ptr.count < RETRANS_MAX) then
          begin
          output TCPM_O.TCPM_MESreq(cur_mes);    (* retrans CC *)
          retrans_ptr.count := retrans_ptr.count+1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          next_state := SAME;
          end;
      else                                          (* count = max *)
          begin
          stop_window_timer;
          stop_inact_timer;
          build_DR;
          output TCPM_O.TCPM_MESreq(cur_mes);
          retrans_ptr.data := cur_mes;    (* replace CC with DR *)
          retrans_ptr.count := 1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          build_DRg;
          output TCPI_O.TCPI_MESreq(tp_mes);  (* send TDISind *)
          next_state := FIN_WAIT;
          end;
     end;



trans
  from ESTAB
    provided (timer(retrans))

     var
          retrans_ptr: retrans_type;

     begin
     retrans_ptr := re_trans.buf[1];
     if (retrans_ptr.count < RETRANS_MAX) then
          begin
          output TCPM_O.TCPM_MESreq(retrans_ptr.data);
          retrans_ptr.count := retrans_ptr.count+1;
          retrans_ptr.timer := RETRANS_TIME;
          set_retrans_timer;
          check_queue;                      (* check for re-trans *)
          next_state := SAME;
```

```
            end;
         else begin                              (* count=MAX *)
            stop_window_timer;
            stop_inact_timer;
            re_trans.total := 1;                 (* remove all retrans *)
            retrans_ptr := re_trans.buf[1];
            build_DR;
            output TCPM_O.TCPM_MESreq(cur_mes);
            retrans_ptr.data := cur_mes;
            retrans_ptr.count := 1;
            retrans_ptr.timer := RETRANS_TIME;
            set_retrans_timer;
            build_DRg;
            output TCPI_O.TCPI_MESreq(tp_mes); (* send TDISind *)
            next_state := FIN_WAIT;
      end;



trans
  provided (timer(retrans))
      from FIN_WAIT

      var
           retrans_ptr: retrans_type;

      begin
      retrans_ptr := re_trans.buf[1];
      if (retrans_ptr.count < RETRANS_MAX) then
           begin
           output TCPM_O.TCPM_MESreq(cur_mes);
           retrans_ptr.count := retrans_ptr.count+1;
           retrans_ptr.timer := RETRANS_TIME;
           set_retrans_timer;
           next_state := SAME;
           end;
       else                                            (* count = MAX *)
           begin
           set_ref_timer;
           next_state := TIME_WAIT;
           end;
      end;
```

```
trans
  provided (timer(window))
    from ESTAB to SAME              (* need to update window info *)

     var
          retrans_ptr: retrans_type;

     begin
     rcv_wnd := set_window;    (* get new credit *)
     build_AK;
     append_flow;              (* append flow-ctrl parameters *)
     output TCPM_O.TCPM_MESreq(cur_mes);
     re_trans.last := re_trans.last + 1;
     retrans_ptr := re_trans.buf[re_trans.last];
     retrans_ptr.data := cur_mes;
     retrans_ptr.count := 0;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     set_window_timer;
     end;



trans
  provided (timer(data_ack))
    from ESTAB to SAME              (* timer for acknowledge of
                                       receiving data *)

     begin
     rcv_ack := rcv_nxt;
     build_AK;
     output TCPM_O.TCPM_MESreq(cur_mes);
     end;



trans
  provided (timer(inact_ack))
    from ESTAB to SAME              (* no DT/ED sent in the interval;
                                       sending AK to keep active *)

     begin
     build_AK;
     output TCPM_O.TCPM_MESreq(cur_mes);
     set_inact_ack_timer;
     end;
```

```
trans
  provided (timer(inact))
     from ESTAB to FIN_WAIT

     var
          retrans_ptr: retrans_type;

     begin
     stop_window_timer;
     stop_retrans_timer;
     build_DR;
     output TCPM_O.TCPM_MESreq(cur_mes);
     re_trans.total := 1;                (* remove all retrans *)
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := cur_mes;
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     build_DRg;
     output TCPI_O.TCPI_MESreq(tp_mes);  (* send TDISind to linker *)
     end;



trans
  provided (timer(ref))
     from TIME_WAIT to CLOSED

     begin
     flag_kill_me := now
     end;
```

```
(***   GROUP-3:  TPDU from TRANSPORT GATEWAY   ***)

trans
  when  TCPI_I.TCPI_MES.ind                  (* CR from gateway *)
    provided (TP_code(message) = CRCODE)
      from CLOSED to SYN_SENTddn

     var
          retrans_ptr: retrans_type;

     begin
     accept_TP_mes (message);             (* take in the TP_mes *)
     rcv_wnd := set_window;
     trans_CRg;
     output TCPM_O.TCPM_MESreq(cur_mes);
     re_trans.total := 1;
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := cur_mes;
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     end;



trans
  when TCPI_I.TCPI_MESind
    provided (TP_code(message) = DRCODE)
      from SYN_SENTddn

     var
          retrans_ptr: retrans_type;

     begin
     accept_TP_mes (message);              (* take in the TP_mes *)
     stop_retrans_timer;
     accept_TP_mes (message);             (* take in the TP_mes *)
     trans_DRg;
     output TCPM_O.TCPM_MESreq(cur_mes);
     re_trans.total := 1;
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := cur_mes;
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     end;
```

```
trans
  when TCPI_I.TCPI_MESind
    provided (TP_code(message) = CCCODE)
     from SYN_SENTout to SYN_RCVDddn

     var
          retrans_ptr: retrans_type;

     begin
     accept_TP_mes (message);          (* take in the TP_mes *)
     rcv_wnd := set_window;
     trans_CCg;
     output TCPM_O.TCPM_MESreq(cur_mes);
     re_trans.total := 1;
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := cur_mes;
     retrans_ptr.count := 1;
     retrans_ptr.timer := RETRANS_TIME;
     set_retrans_timer;
     end;


trans
  when TCPI_I.TCPI_MESind
    provided (TP_code(message) = DRCODE)
      from SYN_SENTout to CLOSED

     begin
     accept_TP_mes (message);          (* take in the TP_mes *)
     trans_DRg;
     output TCPM_O.TCPM_MESreq(cur_mes);
     flag_kill_me := now;
     end;


trans
  when TCPI_I.TCPI_MESind
    provided (TP_code(message) = DRCODE)
      from SYN_RCVDddn to FIN_WAIT

     var
          retrans_ptr: retrans_type;

     begin
     stop_retrans_timer;
     accept_TP_mes (message);          (* take in the TP_mes *)
     trans_DRg;
     output TCPM_O.TCPM_MESreq(cur_mes);
     retrans_ptr := re_trans.buf[1];
     retrans_ptr.data := cur_mes;
```

```
    retrans_ptr.count := retrans_ptr.count+1;
    retrans_ptr.timer := RETRANS_TIME;
    set_retrans_timer;
    end;



trans
  when TCPI_I.TCPI_MESind
    provided (TP_code(message) = DTCODE)
      from SYN_RCVD to ESTAB

    begin
    stop_inact_ack_timer;
    accept_TP_mes (message);          (* take in the TP_mes *)
    trans_DTg;
    output TCPM_O.TCPM_MESreq(cur_mes); (* send DT to TCP Net *)
    retrans_ptr := re_trans.buf[1];
    retrans_ptr.data := cur_mes;
    retrans_ptr.count := retrans_ptr.count+1;
    retrans_ptr.timer := RETRANS_TIME;
    set_retrans_timer;
    set_window_timer;
    set_inact_timer;
    set_inact_ack_timer;
    end;



trans
  when TCPI_I.TCPI_MESind
    provided (TP_code(message) = DRCODE)
      from ESTAB to FIN_WAIT

    var
         retrans_ptr: retrans_type;

    begin
    stop_window_timer;
    stop_inact_timer;
    stop_retrans_timer;
    accept_TP_mes (message);          (* take in the TP_mes *)
    trans_DRg;
    output TCPM_O.TCPM_MESreq(cur_mes);
    retrans_ptr := re_trans.buf[1];
    retrans_ptr.data := cur_mes;
    retrans_ptr.count := retrans_ptr.count+1;
    retrans_ptr.timer := RETRANS_TIME;
    set_retrans_timer;
    end;
```

```
trans
  when TCPI_I.TCPI_MESind
    provided (TP_code(message) = DTCODE)
     from ESTAB to SAME

     begin
     stop_window_timer;
     stop_inact_timer;
     accept_TP_mes (message);          (* take in the TP_mes *)
     trans_DTg;
     output TCPM_O.TCPM_MESreq(cur_mes);
     retrans_ptr := re_trans.buf[re_trans.last];
     retrans_ptr.data := cur_mes;
     retrans_ptr.count := retrans_ptr.count+1;
     retrans_ptr.timer := RETRANS_TIME;
     re_trans.last := re_trans.last + 1;
     set_retrans_timer;
     set_window_timer;
     set_inact_timer;
     end;
```

REFERENCES

[BENHA 83] Benhamou, E., and J. Estrin, "**Multilevel Internetworking Gateways: Architecture and Applications**", IEEE Computer Magazine, Vol.16, pp.27-34, Sept. 1983

[BERKO 87] Berkowitz, Howard C., "**OSI: Whom Do You Trust?**", MILCON '87: Proc. of the IEEE Military Communications Conference, pp.361-369, Washington D.C., October 1987

[BERTS 87] Bertskas D., and R. Gallager, Data Networks, Prentice-hall, New Jersey, 1987

[BOND 87] Bond, John, "**Parallel-Processing Concepts Finally Come together in Real Systems**", Computer Design, pp. 51-74, June 1987

[BRADEN 87] Braden, R., and J. Postel, "**Requirements for Internet Gateways**", RFC 1009, June 1987

[BRUSIL 87] Brusil, Oaul J., and Lee LaBarre, "**Integrated Management of DoD and ISO Networks**", MILCON '87: Proc. of the IEEE Military Communications Conference, pp.34-38, Washington D.C., October 1987

[BURG 84] Burg, F.M., C.T. Chen, and H.C.Folts, "**Of Local Networks, Protocols, and the OSI Reference Model**", Data Communication, pp.129-150, Nov. 1984

[CERF 83] Cerf, V., and E. Cain, "**The DOD Internet Architecture Model**", Computer Networks, October 1983

[CCITT X.200] CCITT, "**Reference Model for Open Systems Interconnection for CCITT Applications**", CCITT Recommendation X.200, 1984

[CHAP 83] Chapin, A.L., "**Connections and Connectionless Data Transmission**", Proceedings of IEEE, Vol.71, pp.1365-1371, Dec. 1983

[CHONG 86] Chong, H.Y., "Software Development and Implementation of NBS Class-4 Transport Protocol", Computer Networks and ISDN Systems, Vol.11, pp.353-365, May 1986

[COLE 86] Cole, Robert, and Peter Lloyd, "OSI Transport Protocol -- User Experience", OPEN SYSTEM '86, pp.33-43

[COMER 88] Comer, D., Internetworking with TCP/IP: Principles, Protocols, and Architecture, Prentice-Hall, New Jersey, 1988

[COURT 86] Courtiat, J.P., A. Pedroza, and J.M. Ayache, "A Simulation Environment for Protocol Specification in Estelle", Proceedings of the IFIP WG 6.1 Fifth Int'l Workshop on Protocol Specification, Testing and Verification, June 1985

[COURT 87] Courtiat, J.P., "How Could ESTELLE Become a Better FDT", Proceedings of the IFIP WG 6.1 Seventh Int'l Conference on Protocol Specification, Testing and Verification, May 1987

[DALTON 87] Dalton, Thomas R., June Downey, and Thomas L. Hahler, "Testing and Evaluation of The Defense Data network", MILCON '87: Proc. of the IEEE Military Communications Conference, pp.398-403, Washington D.C., October 1987

[DAY 83] Day, John D., and Hubert Zimmerman, "The OSI Reference Model", Proceedings of the IEEE, Vol.71, No.12, December 1983

[DEMJ 87] Demjanenko, Victor, and Michael L. Craner, "Simulation of a Distributed Communications Network Using a Multi-tasking Uniprocessor", MILCON '87: Proc. of the IEEE Military Communications Conference, pp.355-359, Washington D.C., October 1987

[FRAN 86] Frankel, Michael S., Charles J. Graff, Larry U. Dworkin, Theodore J. Klein, and Richard L. desJardins, "An Overview of the Army /DARPA Distributed Communications and Processing Experiment", IEEE Journal on Selected Areas in Communications, Vol.SAC-4, No.2, pp.207-215, March 1986

[GREEN 86] Green, Paul E. Jr., "Protocol Conversion", IEEE Transaction on Communications, Vol.COM-34, No.3, pp.257-

268, March 1986

[GROEN 86] Groenbraek, Inge, "Conversion between the TCP and ISO Transport Protocols as a Method of Achieving Interoperability between Data Communications Systems", IEEE Journal on Selected Areas in Communications, Vol.SAC-4, No.2, pp.288-296, March 1986

[HINDEN Hinden, Robert, Jack Haverty, and Alan Sheltzer, "The DARPA Internet: Interconnecting Heterogeneous Computer Networks with Gateways", IEEE Computer, Vol.16, No.9, pp.38-48, September 1983

[ISO 7498] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Basic Reference Model", IS 8072, 1984

[ISO 8072] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Transport Service Definition", IS 8072, June 1986

[ISO 8073] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Connection Oriented Transport Protocol Specification", IS 8073, July 1986

[ISO 8073/DAD2] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Connection Oriented Transport Protocol Specification - Addendum 2: Class Four Operation over Connectionless Network Service", ISO 8073/DAD2, July 1987

[ISO 8208] ISO/TC97/SC6, "Information Processing Systems - X.25 Packet Level Protocol for Data Terminal Equipment", DIS 8208, July 1985

[ISO 8326] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Basic Connection-Oriented Session Service Definition", IS 8326, 1984

[ISO 8327] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Basic Connection-Oriented Session Protocol Specification", IS 8327, 1984

[ISO 8348] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Network Service Definition", IS 8348, April 1987

[ISO 8473] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Protocol for Providing the Connectionless-mode Network Service", DIS 8473, May 1987

[ISO 8648] ISO/TC97/SC6, "Information Processing Systems - Open Systems Interconnection - Internal Organization of the Network Layer",DIS 8648, May 1987

[ISO 8878] ISO/TC97, "Information Processing Systems - Data Communications - Use of X.25 to Provide the OSI Connection-mode Network Service", IS 8878, Sept. 1987

[ISO 8880] ISO/TC97, "Information Processing Systems - - Protocol Combinations to Provide and Support the OSI Network Service", DIS 8880, June 1987

[ISO 8881] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Use of the X.25 Packet Level Protocol in Local Area Networks", IS 8881, ???

[ISO 9068] ISO/TC97/SC6, "Information Processing Systems - Provision of the Connectionless Network Service Using ISO 8208", DP 9068, April 1986

[ISO 9074] ISO/TC97, "Information Processing Systems - Open System Interconnection - Estelle (Formal Description Technique Based on an Extended State Transition Model", DIS 9074, August 1987

[ISO/TR 8509] ISO/TC97, "Information Processing Systems - Open Systems Interconnection - Service Conventions", ISO/TR 8509, November 1985

[ISRAEL 87] Israel, Jay E., and Alan J. Weissberger, "Communicating between Heterogeneous Networks", Data Communications, pp.215-235, March 1987

[LAM 87] Lam, Simon S., "Protocol Conversion --Correctness Problems", Computer Communications Review, Vol.16, No.3, pp.19-29, August 1987

[LAND 86] Landweber, L.H., D.M. Jennings, and I. Fuchs, "Research Computer Networks and Their Interconnection", IEEE Communication Magazine, Vol.24, pp.5-17, June 1986

[LEINER 85] Leiner, Barry M., Robert Cole, Jon Postel, and David Mills, "The DARPA Internet Protocol Suite", IEEE Communications Magazine, Vol.23, No.3, pp.29-34, March 1985

[LINN 85] Linn, R.Jr., "The Features and Facilities of ESTELLE", Proceedings of the IFIP WG 6.1 Fifth Int'l Workshop on Protocol Specification, Testing and Verification, June 1985

[MART 87a] Martinez, Ralph A., ChangWon Son, and JianYi Tao, "Interconnection of SYTEK LocalNet 20 Networks through the Defense Data Network Using Internet Protocol Gateways", Proc. of the IEEE Phoenix Conference on Computers and Communications, Phoenix, AZ, February 1987

[MART 87b] Martinez, Ralph A., "A Look at Internet Gateway Functional Requirement for Tactical Distributed C3 Systems", MILCON '87: Proc. of the IEEE Military Communications Conference, pp.361-369, Washington D.C., October 1987

[MART 88] Martinez, Ralph A., and Changwon Son, "Functional Description and Formal Specification of a Generic Gateway", CERL Technical Report, Computer Engineering Research Laboratory, Electrical and Computer Engineering Department, the University of Arizona, August 1988

[McCOY 87a] McCoy, Wayne, "Military Supplement to the ISO Transport Protocol", RFC 1007, June 1987

[McCOY 87b] McCoy, Wayne, "Implementation Guide for the ISO Transport Protocol", RFC 1008, June 1987

[MILSTD 1777] Military Standard 1777, "Internet Protocol", MIL-STD-1777, RFC 791, September 1981

[MILSTD 1778] Military Standard, "Transmission Control Protocol", MIL-STD-1778, RFC 793, September 1981

[NRC 85] National Research Council, "Transport Protocols for Department of Defense Data Networks", RFC 942, February 1985

[OHARA 87] Ohara, Y., S. Yoshitake, and T. Kawaoka, "Protocol

Conversion Method for Heterogeneous Systems Interconnection in Multi-Profile Environment", Proceedings of the IFIP WG 6.1 Seventh Int'l Conference on Protocol Specification, Testing and Verification, May 1987

[OKUM 86] Okumura, Kaoru, "A Formal Protocol Conversion Method", ACM SIGCOMM '86: Symposium on Communication Architectures and Protocols, August 1986

[PARDUE 87] Pardue, Mark D., "Fine-Tuning the OSI Model: Layer Functions and Services", MILCON '87: Proc. of the IEEE Military Communications Conference, pp.199-203, October 1987

[PIATK 86] Piatkoski, Thomas F., "The State of the Art in Protocol Engineering", ACM SIGCOMM '86: Symposium on Communications Architectures and Protocols, August 1986

[POST 80a] Postel, J., "Internetwork Protocol Approaches", IEEE Transaction on Communications, COM-28, pp.604-611, April 1980

[POST 80b] Postel, J., "User Datagram Protocol", RFC 768, August 1980

[POST 81] Postel, J., "Internet Control Message Protocol", RFC 792, September 1981

[ROSE 87] Rose, Marshall T., Dwight E. Cass, "ISO Transport Services on Top of the TCP", RFC 1006, May 1987

[RUDIN 85] Rudin, Harry, "An Informal Overview of Formal Protocol Specification", IEEE Communication Magazine, Vol.23, No.3, pp.46-52, March 1985

[SIRBU 85] Sirbu, Marvin A., and Laurence E. Zwimpfer, "Standards Setting for Computer Communication: the Case of X.25", IEEE Communications Magazine, Vol.23, No.3, pp.35-45, March 1985

[STALL 88] Stallings, W., Data and Computer Communications, 2nd Edition, Macmillan, New York, 1988

[TANEN 88] Tanenbaum, A.. Computer Networks, 2nd Edition, Prentice Hall, New Jersey, 1988

[TSUCH 87] Tsuchiya, Paul F., "**Team-of-Gateways: Design and Implementation**", MILCON '87: Proc. of the IEEE Military Communications Conference, pp.39-42, Washington D.C., October 1987

[VENK 85] Venkatraman, R.C., and T.F. Piatkowski, "**A Formal Comparison of Formal Protocol Specification Techniques**", Proceedings of the IFIP WG 6.1 Fifth Int'l Workshop on Protocol Specification, Testing and Verification, June 1985

[WEISS 87] Weissberger, Alan J., and Jay E. Israel, "**What the New Internetworking Standards Provide**", Data Communications, pp.141-156, February 1987