A COMPUTER NETWORK: STRUCTURE AND PROTOCOLS OF THE RPCNET

Fausto Caneschi

December 1978

PREFACE

This manual was written with a well defined goal: to collect all the technical specifications and protocols of the RPCNET, the Italian Computer Network. Since now, these technical specifications were described in a number of papers, and most of them were obsolete. In fact, one thing is a new protocol before its implementation, and,very often, different after the implementation. Anyway, the principles on which RPCNET was established remained untouched, so that the author's work was made easier by consulting and using the material contained in those old papers.

Nothing could be done, however, without the help of the RPCNET people, both at CNUCE and at the IBM scientific center of Pisa, Italy. My special thanks go to Luciano Lenzini, Fabio Tarini, Claudio Menchi, Marco Sommani, Maurizio Martelli and Erina Ferro at CNUCE, and to Livio Lazzeri, Alessandro Fusi, Carlo Paoli and Raffaello Porinelli at the IBM scientific center.

The author admits that this work can be largely improved and thus will appreciate all the suggestions and contributions, both from the above mentioned specialists and from any other interested reader.

ABSTRACT

A brief description of the RPCNET architecture is given in the beginning of this manual, then, the protocols and the packet formats of RPCNET are described.

More precisely, Chapter 1 deals with a general description of the RPCNET, Chapter 2 deals with the 1st level protocol (Line and Reconfiguration protocols), Chapter 3 deals with the 2nd level protocol (End-to-End protocol), and Chapter 4 deals with User-Level protocols, including the description of RNAM, the generalized Access Method supplied by RPCNET.

Appendix A is the hardware scheme of the BSC-modified line connection. Appendix B gives an example of how the reconfiguration protocol works, and finally Appendix C describes the packet formats.

LIST OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

This technical manual describes the various layers and protocols of RPCNET, the italian computer network, and is intended for computer personnel who wish to implement the RPCNET architecture on their own existing systems. Although the RPCNET architecture is independent on the computing systems on which it can be implemented, the original partners of the REEL project (REte di ELaboratori) all employed IBM computing systems, so that the RPCNET software for IBM 370s, running OS/VS2, MVS or VM/370, and for IBM S/7 was developed and tested in the course of the project. This software is available at CNUCE, C.N.R., via S. Maria 36, Pisa, Italy.

2 RPCNET ARCHITECTURE

Although it is assumed that the reader of this manual is already familiar with the RPCNET architecture,(1), (2), (3), (4), a brief description of it follows here as a background for the material presented in subsequent chapters.

2.1 General Information.

RPCNET is a packet switching, distributed control, computer network; automatic re-configuration and a general Access Method are distinguishing characters of its design.

RPCNET is logically divided into three layers:

a) The Communication Layer

b) The Interface Functions Layer

c) The Application Layer,

which correspond to the three addressing levels of the network.

2.2 The Communication Layer

One function of the Communication Layer is the sending of packets on the line(s) according to a well defined line protocol. It also performs the routing and re-configuration functions, using the NETCHANGE protocol.

The software component which pertains to this layer is the Common Network Manager (CNM), which is the first addressable unit in RPCNET. This means that a network address like nØØ (where n is an integer number between 1 and 65536) is a CNM address.

Any computer on which the Communication Layer is implemented is considered to be an RPCNET Node, and as such it can perform the line-driving and the re-configuration functions. The packets are required to be fixed-length, formatted fields ,the RPCNET packet length being 255 bytes.

2.3 The Interface Functions Layer

The Interface Functions Layer facilitates the network users' (Applications) work, by:

a) hiding the packeting activity, the retransmissions and the reconfigurations;

b) handling the logical RPCNET channel, notifying the Application in case of errors.

The Interface Functions Layer is composed of two main modules: the Session Handler (SH) and the Network Services Manager (NSM). The SH is in turn composed of 3 modules and performs the in-session functions of assembling and disassembling the application buffers (Basic Information Unit or BIU) in packets for the Comm_nication Layer. The SH has a window oriented protocol, which allows the discarding of duplicate packets and the recognition of BIU loss. The 3 SH modules are designated as: the Presentation Services, the Data Length Adapter and the Data Flow Control, which are described later in this manual.

The NSM regulates the opening and closing of the Logical Channel and provides services as the transmission of "mail".

All the NSMs in the network are connected by means of a NSM protocol which has a fixed length window. The presence of an NSM characterizes a computer as a RPCNET Host, making the NSM as the second addressable unit in the network. A network address such as nmØ (where n is the Node CNM address, and m is the Host NSM address) designates the m-th Host in the n-th Node.

The Logical RPCNET Channel mentioned previously as being handled by the Interface Functions Layer, has the following characteristics:

a) it is driven with an half-duplex technique

b) it does not provide an error-free connection

c) it detects and signals, at both Logical Channel sides, errors in the form of BIU losses.

However, with little effort the facilities of the Logical Channel could be extended by making it a full-duplex connection. This should be kept in mind during the implementation phase.

2.4 Application Layer

In the RPCNET, the term Application is used to indicate the generic network end-user: an Application is defined as any process or set of cohordinated processes, that access the Communication System (Communication Layer and Interface Functions Layer) in order to obtain network services. An Application is identified by the third address field in the network address.

A network address such as nmk identifies the k-th Application of the m-th Host in the n-th Node. In the network address space, the nmk unit is known as the nmk LCT (Logical Channel Termination).

3 THE COMMUNICATION LAYER

3.1 General Information

The Communication Layer is composed of three modules:

a) The Packet Switcher (PS)

b) The Communication Network Manager (CNM)

c) The Network Connection Handler (NCH)

It must be noted, at this point, that a RPCNET connection is a full-duplex channel: this means that the NCH is in charge of providing a full-duplex connection for its users (namely, the PS and the CNM), and to hide to them the details of the protocol(s) used. As the initial RPCNET partners have IBM computers, a BSC full-duplex protocol (2) was designed and implemented which allows, by means of a special "Y" cable (see Appendix A), to use a BSC line as a full-duplex line.

If in the decision phase of the project the BSC line will not be chosen, the line protocol must be re-designed.

In the first section of this chapter we will describe the PS; the second section is dedicated to the CNM, and the third to the BSC full-duplex NCH.

3.2 The Packet Switcher

The PS is in charge of routing packets towards their destination, using the Routing Table, which is maintained and updated by the CNM (see 3.3). The PS has an input queue, which is used by:

a) the upper level of the network (SH or NSM)

b) the NCH

For every packet in the input queue, the Destination Address Field (DAF) is tested, and the packet is:

a) enqueued on the NCH input queue corresponding to that address (as indicated in the Routing Table)
or

b) enqueued on the upper level input queue

The PS activity can be sketched in a flowchart, see Figure 1.

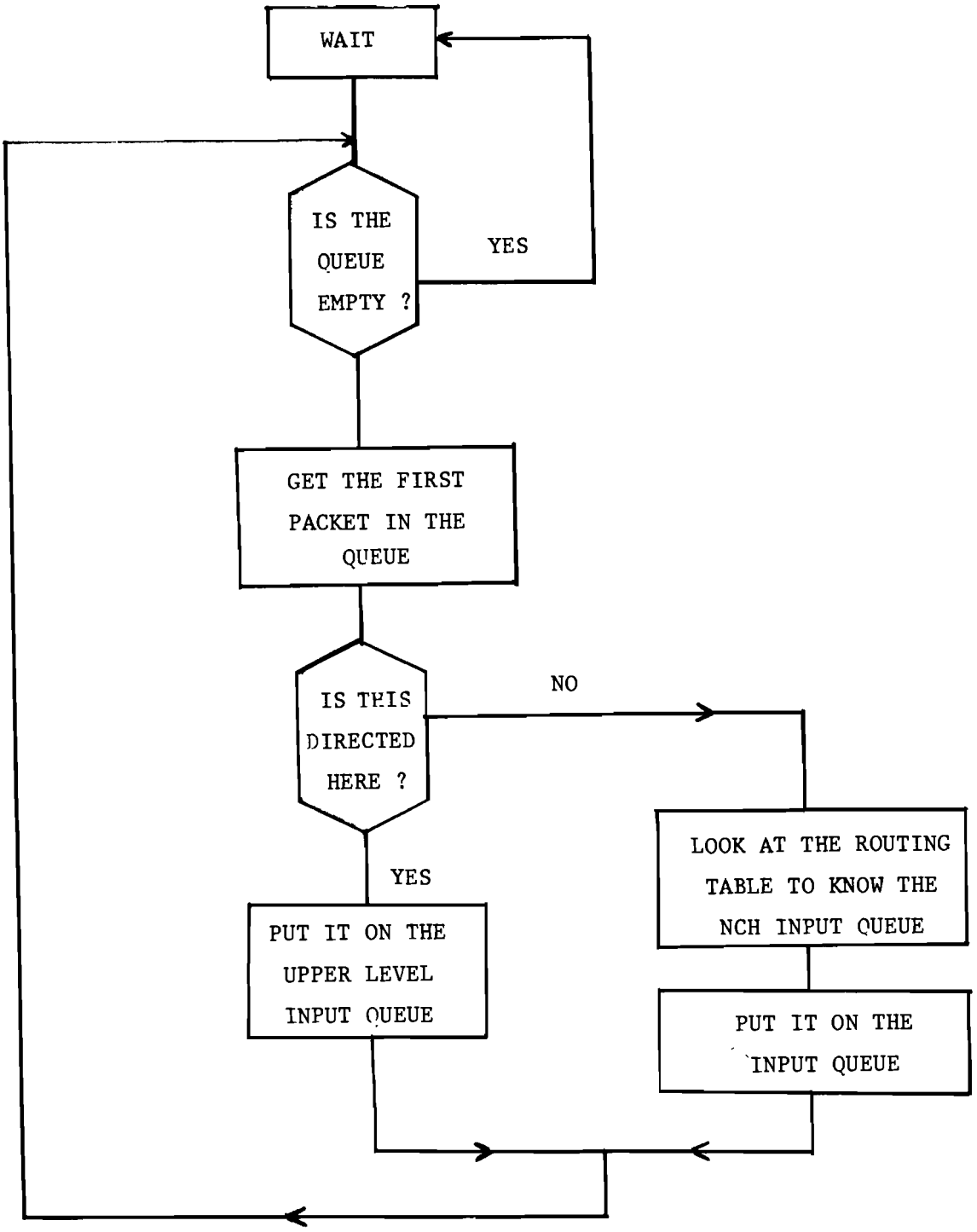


Figure 1 Packet Switcher activity

3.3 The Communication Network Manager

The CNM is a component which keeps up-to-date those tables which are necessary for describing the network topology (Distance Table) and also for deciding to which adjacent node a packet must be forwarded to, in order to reach its destination (Routing Table).

The CNM is also in charge of preparing and sending reconfiguration messages (used to make the nodes of the network aware about the topology changes).

The only other component which interacts with the CNM is the NCH. The NCH makes the CNM aware about the going down and the coming up of the links it controls: in addition, it gives to the CNM the reconfiguration messages it receives, and sends over the links the reconfiguration messages prepared by the CNM.

We will describe the protocol used by the CNM (the NETCHANGE protocol) in the following section: this protocol is introduced in (2) and a proposal for the introduction of the traffic load, defining the VRP (Virtual Reconfiguration Protocol) was presented in (3).

The VRP is not implemented in this version of RPCNET, but it is fully compatible with NETCHANGE, so that the implementer should know it, in view of a possible introduction of it in RPCNET.

3.3.1 The NETCHANGE protocol

In the actual implementation of the RPCNET, the NETCHANGE protocol has been a little modified, in order to gain in speed: the modification consists on sending for every reconfiguration message, the entire Routing Table (only the distance column, of course) and not the modified row only.

All the terms which are not well defined now will be explained in the following. For an implementation example see (4).

The routing algorithm performed by the NETCHANGE protocol is a deterministic one: in fact, it computes the best route to reach a node by taking into account only the topology of the network; a route can change only if the topology changes.

The NETCHANGE protocol assumes that the routing algorithm is based on the shortest distance criterion, i.e. packets in transit are returned towards the adjacent node which is located on the shortest path to their destination. Such a routing algorithm does not need to know the entire topology of the network; it is sufficient to know the distance between the adjacent nodes and all the other nodes in the network.

Let us assume, for instance, that the node A has to send a packet to the node X, not adjacent to A. If B is an adjacent node, all that A needs to know is the distance between B and X: the distance from A to X via B can be computed as the distance between B and X plus one, so that A will route the packet to the neighbour declaring the shortest distance.

The algorithm requires two tables: the Distance Table and the Routing Table. The Distance Table has a column for every neighbour and a row for every node in the network. The Routing Table has an entry (row) for every node in the network, which contains the shortest distance to reach this node, and the name of the first node in the route (the neighbour to which the packets are to be sent for that path).

An example of a network topology, with the Distance Table and the Routing Table for a node, is given in Figure 2.

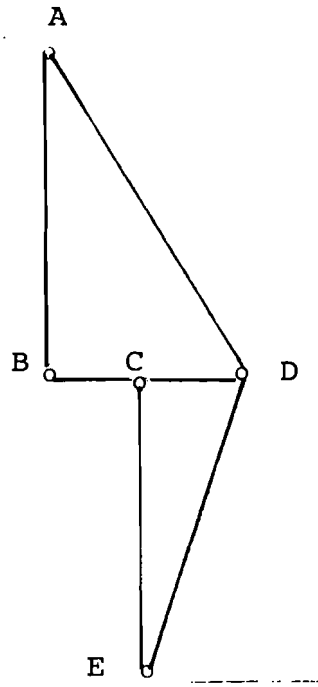
The Distance and the Routing Tables are present in every node, they also contain a row for each node in the network. The Distance Table has as many columns as the links going out from its node; every column has, for every row, the distance between the node the table refers to and the node corresponding to the row.

The Route Table has two columns, referring to:

1 The shortest distance

2) The node to which messages are routed

The Routing Table for NETCHANGE is determined in the following way: at the beginning, the Routing Table is deduced from the Distance table by simply selecting the minimum distance in each row. Later, messages are exchanged from node to node, carrying information about the network's topology. They have the form of the Routing Table of the node sending the message.



	CB	CD	CE		
A	2	2	3	2	B
B	1	3	3	1	B
C	-	-	-	-	-
D	3	1	2	1	0
E	3	2	1	1	E

Distance (left part)
and Routing Tables for node C

Figure 2 Distance and Routing tables for node C

In a node, one of the following events can occur:

1 an adjacent link comes up;

2) an adjacent link goes down;

3) a reconfiguration message is received.

These three events correspond to three different algorithms that must be performed by the node. To illustrate the algorithms, the following notations are used:

A is the node carrying out the algorithms

B is a neighbor of A

C is any other node different from A and B

N is the number of the nodes in the network

The number N has a special meaning for the protocol; because the longest path between any two nodes cannot pass through more than N-1 different nodes, the protocol makes the following assumptions:

if N is contained in any field of the Distance Table, the node corresponding to the row in which it is located cannot be reached passing through the link represented by the column;

if N is contained in the Routing table the node cannot be reached at all (for example, the node is down).

Algorithm 1

An adjacent link AB (or a neighbor node B) comes up.

1 The entry, row B, column AB of the Distance table is set to 1.

2) Then, row B of the Routing table is set to 1,B (distance column 1, node column B)

3) A copy of A's Routing table (distance column) is sent to all the neighbors of A (including B).

Algorithm 2

An adjacent link AB (or a neighbor B) goes down.

All the entries in the column AB of the Distance Table are set to N; then, for each node X except for A do:

1 Examine row X of the Distance table; calculate the new shortest distance (procedure CALC); if it is not different, end for X

2) If it is, make the appropriate changes to the entry X of the Routing table, and send a reconfiguration message to all the neighbors

Algorithm 3

A reconfiguration message is received

Let K the new value for the distance of the node X: for every row of the Distance Table except for A:

1 Entry X of the Distance Table is set to MIN(K+1,N)

2) The new shortest distance for X is calculated (CALC)

3) If any row of the Routing table has been changed, send it to all the neighbors (reconfiguration message).

Procedure CALC

All the entries in the table refer to row X

1 For all the links (Distance Table columns) which do not have the distance equal to N, the distance is picked up

2) If the set is empty, the node X cannot be reached, and the Routing Table entry is changed to 'no route' (N).End.

3) If the set is not empty, the smallest elements are chosen; from these, the first distance on the Distance Table from left to rigth is selected. The corresponding link is placed in the route part of the Routing table.

An example is given in Appendix B, using the topology depicted in Figure 2.

3.4 The Network Connection Handler (NCH)

As explained before, since now the RPCNET implementation made use of BSC lines, so that a BSC full-duplex protocol was designed and implemented for the NCH. For what follows, the reader is assumed to have a good knowledge of the BSC control characters, and also of the standard BSC protocol in general.

3.4.1 General considerations

In order to take advantage of the procedure ability to transmit a stream of data in both directions at the same time, an overlapping between the block acknowledgements and the transmission of a subsequent block is provided. Generally speaking, it is possible to say that as soon as one block is transmitted as a result of a write operation, the next block is sent, regardless of whether or not the previous transmission was completed without error.

According to this protocol, every block of information is shipped out by the computer on the line as a transparent text. It will be assumed that all padding and frame characters are provided implicitly; it is not important whether from hardware and/or software. Transmission checking is provided in the form of CRC when the normal end of transmission is detected.

According to the protocol, each physical line is broken up into a number of logical "channels", in the present implementation eight channels in each direction.

Negative (NACK) and positive (ACK) acknowledgments required by a BSC transmission are now replaced by a control message BSCM which includes both positive and negative acknowledgments. All new definitions will be given below. The control message BSCM is shipped out onto the line together with a text, if there is one; consequently, BSCM uses the same framing characters as the text does. A BSCM is always referred to one channel, but it carries a string of bits (a byte called ACKM) describing the binary state (ACK or NACK) of all the receiving channels. It follows that BSCM consumes an insignificant portion of line band width at times of heavy load. Both BSCM and text carry several bits of control information as will be described later.

Before describing how the protocol works, the following conventions are introduced:

- a one to one correspondence between the transmitting channels and the eigth sections of a circle;

- a one to one correspondence between the eigth bits of a byte and the receiving channels, as it is possible to see in Figure 3.

- a transmitting channel can be either in a "busy" (b) or in a "free" (f) state, depending on whether or not it is currently associated with a block.

- Further, a receiving channel can be in a no error (g) or in an error (b) reading state depending on whether or not a block which refers to that channel has been received correctly.

Chan 0 1 2 3 4 5 6

Figure 3 Logical Channels and Byte Structure

3.4.2 How the protocol works

Each block going in one direction is associated with a name and transmitted onto a channel. While the association between block and name is made in a very naive way, the way in which a link between block and channel is made is more complicated, and has to be explained in details. The following strategy is used: blocks are transmitted on sequential "free" channels in order to keep track of the time order of the transmit operation. As soon as one block is transmitted, the corresponding channel is marked busy. The state of the channel is changed only when a BSCM is received from the other side carrying information connected with that channel.

At the receiving side, blocks are expected in the same channel order as they were sent. Any time that this order is

broken, it is logical to require the re-transmission(s) of the block(s), associated to the channel(s), which has (have) been received out of sequence.

For example, let's assume that computer S sends blocks to computer R named a, b, and c on the channels m-2, m-1 and m respectively. According to the graphic representation above, the situation in S will be what we can see in Figure 4.

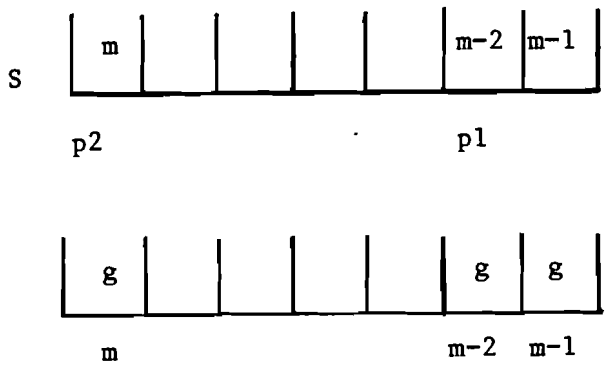


Figure 4 Example of a Sender Node

Channels between pointers p1 and p2 will be associated to blocks that have already been transmitted but for which no BSCM has yet been received.

In the example just mentioned, p2 will point to channel m and p1 to channel m-2, if no BSCM has yet been received after the transmission of m. At the receiving end, if no errors occur, blocks a, b, and c, on the channels m-2, m-1 and m respectively, will be received sequentially. BSCMs will be sent back to S, reporting for the receiving channels of R the situation in Figure 5.

Presuming that S receives the BSCMs back correctly, channels m-2, m-1 and m will be set "free" gradually one by one. Thus, they can be re-used for other block transmissions. In terms of pointers, p1 will tend to p2.

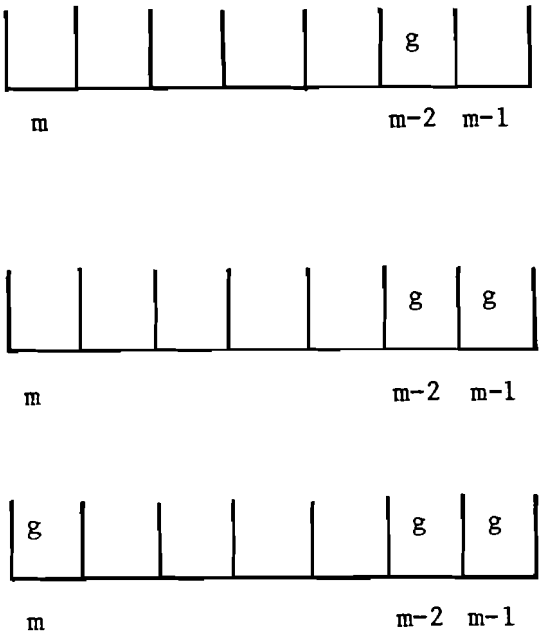


Figure 5 Example of a Receiver Node

It is worthwhile emphasizing that, due to the ACKM byte in the BSCM structure, it is not necessary that 3 BSCMs be returned to S. The most recent BSCM is capable of describing the state of the channels up to the moment of its construction.

In this example, if R receives a BSCM belonging to the m-th channel, R is capable of reaching the same final result that it should be achieved if it had received one BSCM at a time, the only difference being the time in which every channel will be set free.

Let's now suppose that R receives correctly blocks a and c belonging to channels m-2 and m respectively. As they are accepted by R, the blocks are properly acknowledged. Assuming that the BSCM corresponding to channel m-2 reaches S, the procedure to be followed at S for channel m-2 is the same as described above. On the other hand, as soon as the block associated to channel m is received, the byte describing the portion ACKM of BSCM will have the structure described in Figure 6.

The channel m-1 will be put into the error reading state because according to the algorithm with which blocks are sent to S, the block connected with channel m-1 should

Figure 6 Example of ACKM Structure

have been received, from the time view point, between the blocks related to m-2 and m respectively. Presuming that S receives the byte shown in Figure 6, the following procedure will be performed:

a) channel m-2 will be set "free" if its BSCM has not yet reached side S;

b) channel m-1 will be set "free" and block b will be transmitted with the same name but on channel m+1 if this channel is free (available);

c) channel m will be set "free"

At the end of these operations we have the situation shown in Figure 7.

Figure 7 Blocks Situation

BSCM is always referred to the channel number of the last block received correctly.

Finally, it is worth noting that the Sender must not have more than 7 blocks outstanding, i.e. blocks transmitted and about which no news have been received (ACK or NACK). This is to avoid confusion in the interpretation of the BSCM message.

3.4.3 Block name management

As specified before, each block going in one direction carries a name with other information. As, at the most, it is possible to have as many different names as the number of channels, names going from Ø up to 7 are needed now.

According to this protocol, the availability of a name is described by a bit which is called availability - unavailability bit. To detect duplicates at the receiving side, another bit, called parity bit, is needed at both sides.

Consequently, the message name space used by this protocol is 16, and it is managed in 8 pairs as follows:

- the unavailability-availability bits indicate whether or not the name has already been used. The parity bit is kept at both sides for each possible name.

- at the receiving side, if the parity bit of the received block does not match the parity bit associated with the appropriate name, the received parity bit is complemented, otherwise the block is a duplicate and discarded.

- when the transmitting side receives a block, the BSCM bits are examinated one by one, and the following procedure will be adopted for each channel marked (g) in the ACKM bits:

a) for each free transmitting channel, no action is taken because only one of these two alternatives is possible:

1 a block has never been transmitted on this logical channel;

2) the channel has been set free by a previous BSCM;

b) for each busy transmitting channel, the corresponding channel is marked free, the name is marked available, and the parity bit is complemented. In other words, both the channel number and the name are available for other block transmissions.

3.4.4 Protocol specification

The block structure actually implemented was defined by keeping in mind the following points as being the most important requirements:

- in order to consume as little line bandwidth as possible BSCM and the text (if there is one) must be transmitted together by making use of the same framing characters.

- due to the channel structure used by this protocol, at any time the number of the blocks waiting for acnowledgment at the transmitting side cannot be higher than the total number of channels (now eight); therefore, ACKM can be represented by one byte.

Once specified that, as far as the BSC transmission technique is concerned, the block structure believed to be the simplest is as in Figure 8,

where:

ACKB (8 bits)

Ø-2 Channel number of last received block

3 Not used

4 Parity indicator of last received block

5-7 Block name of last received block

ACKM (8 bits)

Acknowledge mask. Bits Ø-7, from rigth to left, refer to the blocks sent immediately before the block identified by ACKB. If the bit is set to 1, the block is in error, if it is set to Ø, the block is positively acknowledged. An example is given in Figure 9.

ACKBC and ACKMC (8 bits)

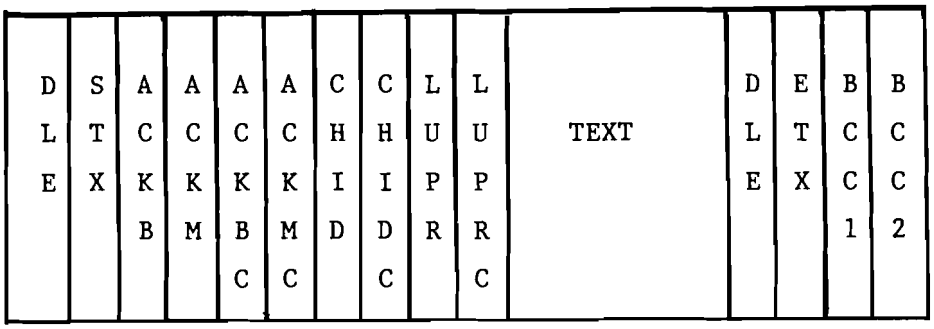


Figure 8 Block Structure

Figure 9 Channel 2 is nacked

These two bytes are the one's complement of ACKB and ACKM, respectively. They are used to verify the validity of ACKB and ACKM, even if the block of which they are part has a CRC error.

CHID (8 bits)

Ø-2 Channel number of the block (if this is a non-local block

3 Local/non-local indicator (see later)

4 Parity indicator (non-local blocks)

5-7 Block name (non-local blocks)

CHIDC (8 bits)

This byte is the one's complement of CHID. It is used to verify the validity of CHID only when the CRC is incorrect.

LUPR (8 bits)

This byte denotes the number of the last received update concerning the Routing table (see later).

LUPRC (8 bits)

This byte is the one's complement of LUPR.

In order to clarify this implementation, the following example is given. Let us assume to have received a block having for BSCM the following bit structure:

ACKB = X'5C' = B'Ø1Ø111ØØ'
ACKM = X'4A' = B'Ø1ØØ1Ø1Ø'

Bits Ø-2 and 5-7 of ACKB define the base block having the channel number 2 and the block name 4.

From the definition of ACKM it follows that (by looking at the ACKM bit pattern from rigth to left and remembering that the received block has been transmitted on channel 2) the block transmitted on channel

1 is negative acknowledged

Ø is positive acknowledged

7 is negative acknowledged

6 is positive acknowledged

5 is positive acknowledged

4 is negative acknowledged

3 is positive acknowledged

3.4.4.1 Connection Maintenance Protocol. The Link status has to be monitored, in order to know at any time if it is active or if it has to be considered down. For this reason, the full-duplex protocol has to be completed with a connection protocol that is able to detect the "going down" and the "coming up" of a Link.

We shall discuss the "coming up" in the next section. Here let us emphasize how the "going down" is detected. As a first approach, we can realize that a Link went down when we try to use it: in such a case, we do not receive any ACK for the blocks we began to send. If we want to take a faster action, we must detect that a Link is down at the time it actually goes down. This aim is achieved by maintaining always a certain amount of bi-directional traffic on the Link. The two adjacent FEPs are always listening to their receiving lines with a certain timeout: if the timeout elapses and no block is received in the meanwhile, the Link is considered down. This means that, if there is no block of data to send, fictitious traffic must be provided. "Hello" messages are prepared specifically for this.

Since the purpose of the Hello messages is to provide the fictitious traffic, they do not necessarely need an acknowledgment. In this way channels and block names would not be used, avoiding system overhead. In addition, a certain kind of acknowledgment would still be provided by the timeout.

The possibility of transmitting blocks outside the first level protocol seems to be useful in many other circumstances. Some other examples are:

- the transmission of ACKs alone (in order to avoid a certain kind of loop of ACKs)

- the initialization of the Link (when the first level protocol is not yet available, etc).

In this view, it is worthwhile to introduce a special type of block, the "local" block. Local blocks are defined as those blocks carrying an ACK and transmitted outside the first level protocol, that is without requiring an acknowledgment. Local blocks do not involve any module other than the NCH, because they are born and they die inside the NCHs. In order to identify this type of block, bit 3 of CHID is used.

3.4.4.2 The Initialization of the Link. It is clear that when a Link is in the inactive state, not even the minimum amount of bi-directional traffic can be mantained. This does not mean that no activity is performed on that

Link. In fact, if a Node has an inactive Link, it must be kept listening to that Link, in order to be able to switch to the active state upon receiption of the appropriate initialization sequence.

The purpose of the initialization sequence is to perform some checks in order to verify if the active state can be entered. Among the points to be checked are:

- the working of both the sending and the receiving line;
- the compatibility of the software running on both the adjacent Nodes
- the Node identification, etc.

In view of this, let us introduce the "status" as the amount of information exchanged between two adjacent Nodes during the initialization. It is clear that in this preliminary exchange of information, the first level protocol is not involved at all.

In order to allow status exchange, let us introduce another type of local block, the "status" message. There are three types of "status" messages:

1 request of status

2) transmission of status

3) request and transmission of status

Whenever an attempt is made to start a Link, a request of status is sent to the adjacent Node. If the request is received correctly, the adjacent Node will answer by supplying its own status. The status information is checked by the local Node, and if it is rigth, a Hello message is sent. The reception of a Hello message is the triggering event: at the receiving side the active state of the Link is entered, and therefore the Hello messages regime is established.

3.4.4.3 Local Block Format. The first four bytes of the block are always the same for every block type. They are: DLE, STX, ACKB, ACKM. Bit 3 of the fifth byte is used to distinguish local and non-local blocks. When bit 3 is on (local block), the byte structure is as follows:

Ø-2 channel number on which the next non-local block will be transmitted

3 local/non-local indication (set to 1)

4 request of synchronization indicator (after start or restart)

5 stop request indicator

6 status request indicator

7 status included indicator

When bits 4,5,6 and 7 are all zero, the local block contains an Hello message.

The status field, when included, contains all the information necessary to initialize the connection maintenance protocol. The status field is placed immediately after the LUPRC byte, see Figure 1Ø.

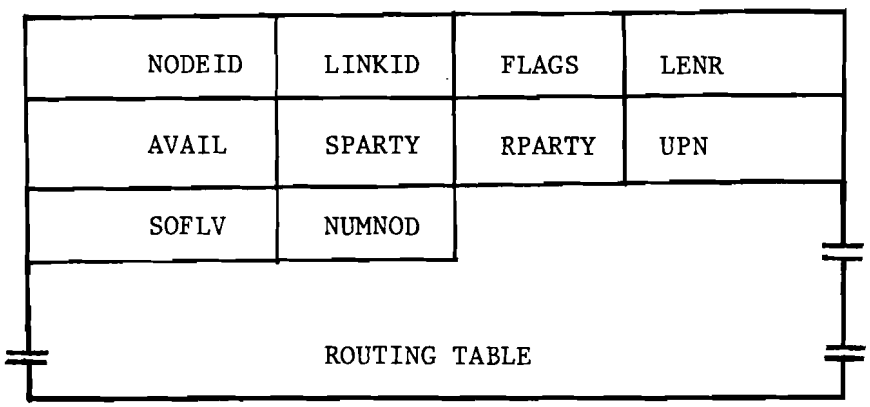


Figure 1Ø Status Message

where:

NODEID is the number identifying the Node of the Sender

LINKID is the number identifying, at the sending side, the Link over which the block is transmitted

FLAGS all reserved
bit Ø: Sending Line Active/Not Active
bit 1: Receiving Line Active/Not Active
bit 2: Link Active/Not Active
bit 3: Link in Prepare State/Not in Prepare State
bit 4: Sending in Progress/No Sending in Progress
bit 5: Receiving in Progress/No Receiving in Progress
bit 6: Writing required/No Writing Required
bit 7: Update Required/No Update Required

LCHR last channel number received correctly

AVAIL availability of the names (a bit set to 1 indicates that the corresponding name is busy)

SPARTY parity of the names for the next transmission

RPARTY parity of the names after the last receiption

UPN routing table update number in this block

SOFLV software level

NUMNOD number of nodes supposed to be in the network

4 THE INTERFACE FUNCTIONS LAYER

4.1 General Information

The Interface Functions Layer provides the interface between the Common Network (Connection Network plus Communication Functions Layer), and the Application Layer, that is the layer in which the Network Users (Netusers) reside.

The Interface Functions Layer basically provides:

a) the defining and undefining of ports (Logical Units) on the Communication System

b) extemporaneous services, such as query and mailing

c) setting up and closing down of Logical Channels between the Applications.

Once established, Logical Channels are maintained by a software component of the Interface Functions, called Session Handler (SH). Along the Logical Channel, Applications exchange units of information, called BIU (Basic Information Unit).

The combination of the Interface Functions Layer and the Common Network constitutes the Communication System.

The characteristics of a Logical Channel can be summarized as follows:

a) it is driven with an half-duplex technique;

b) it does not provide an error-free connection;

c) error (loss of BIU) is signalled at both ends of the Logical Channel.

It is worth noting at this point that a possible, little effort improvement of the characteristics of a Logical Channel could be a full-duplex protocol. One must keep this in mind, when implementing RPCNET, to reserve this possibility for the future.

A third component of the Interface Functions Layer is the CALL/RETURN interface (in the VM and OS implementations

called NAC, Network Access Controller), which performs a number of tests on the operations issued by the user (the Application) before passing the request to the other modules.

In the following we will describe in detail the operations and the protocols of the three components of the Interface Functions Layer, namely:

a) the Session Handler (SH)

b) the Network Services Manager (NSM)

c) the Network Acces Controller (NAC), or CALL/RETURN Interface.

In Appendix C the formats of the packets used by the NSM and the SH are illustrated.

4.2 The Session Handler

The Session Handler (SH) provides a mechanism by which two Applications can communicate each other during a session.

4.2.1 Operation

The SH receives, from an Application, units of information called Request Units (RU), together with additional control information. The SH uses the control information to build a Request Header (RH), which is prefixed to each RU sent and which is interpreted by the SH receiving the RH-RU combination. This combination is called BIU (Basic Information Unit).

During a session, the two Applications are connected through a Logical Channel, which, in turn, can be considered as an half- duplex connection, with these characteristics:

a) no error free connection

b) error detection and signalling at both sides of the Logical Channel

c) RU limited in size

d) RU data integrity maintained

e) RU sequentiality maintained

f) RU duplication impossible

These characteristics are to be mapped on the Common Network characteristics, which are:

a) packet length limited

b) order of packets not maintained

c) loss or duplication of packets possible

d) packet data integrity preserved

e) changes in the Common Network connectivity notified.

4.2.2 Internal Protocols

In order to build the LC characteristics on the Common Network, the SH has to use some internal protocols, which are completly transparent to the interconnected Applications.

Because of the complex SH structure, some functional units can be distinguished, as show in the functional scheme of Figure 11.

4.2.2.1 Application The two Applications exchange data (RU) through the Logical Channel following the LC rules (see RNAM macros). Because of the possibility of RU loss, the Applications have to provide an agreement upon the recovery. The type of the recovery depends on the Application characteristics, and should be defined at that level.

4.2.2.2 Presentation Services. In order to facilitate the Applications about the LC usage some presentation Services are provided:

a) handling of the Flip-Flop driving of the LC;

b) providing a logical data entity considered at the Application level as an unit, set up by an RU chain;

c) providing three types of response on the single LC operation: no-response, exception response and definite response.

4.2.2.3 Data Length Adapter. Because of the unrelated data length limitations of the LC, and of the Packet Switching System, an RU data length adaptation becomes necessary. For this reason, a segmenting/reassembling service has to be provided.

The unit of information destinated to the packet Switching System and built up with a BIU or BIU segment prefixed with addressing and reconstruction information (TH, Transmission Header), is called Path Information Unit (PIU). So that, this SH section, besides providing the BIU segmenting/reconstruction, has to preserve the BIU data integrity, discarding the BIU that cannot be completed, and notifying the related exception.

4.2.2.4 Data Flow Control. In order to realize the LC characteristics, the Data Flow Control has to provide:

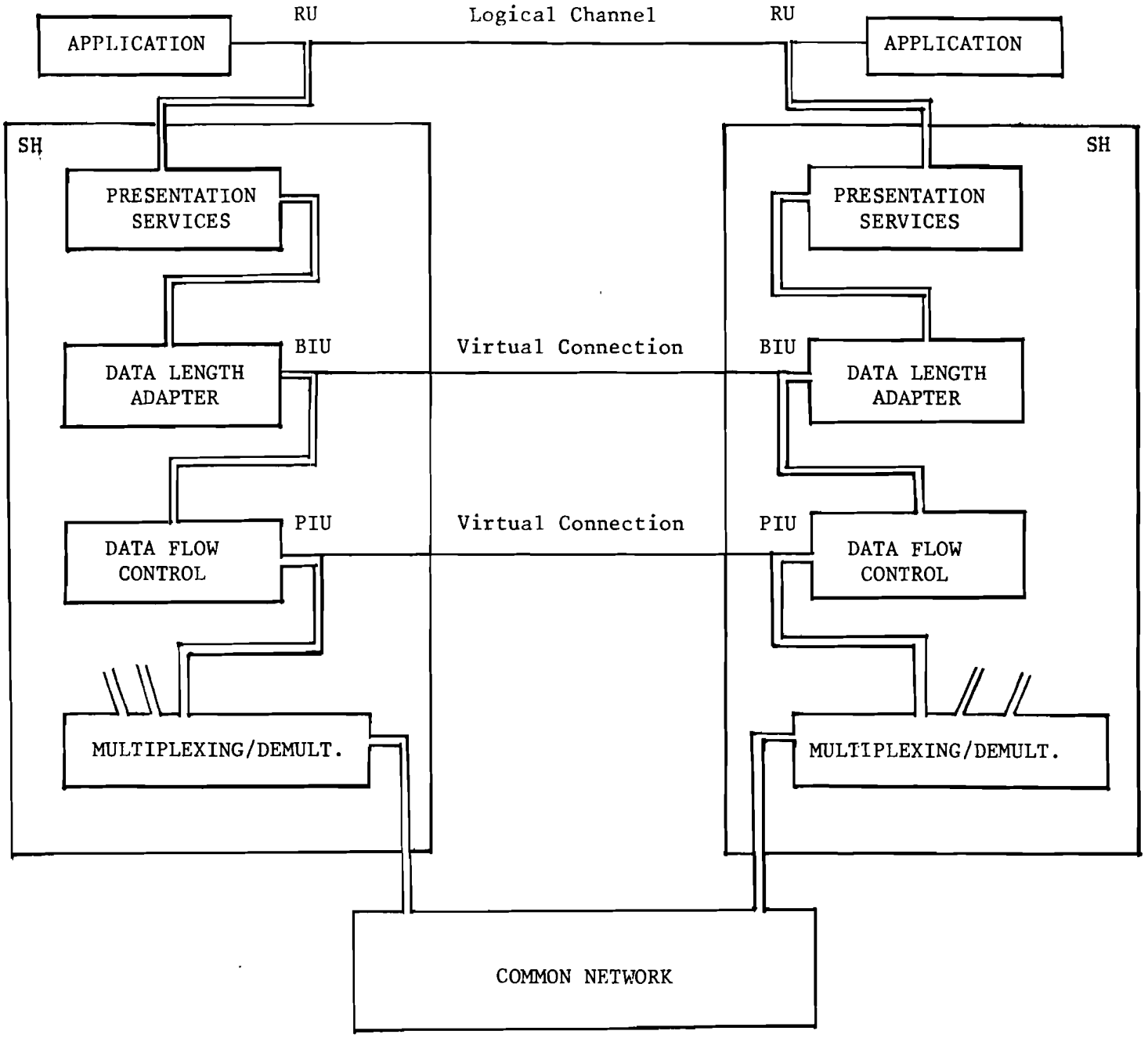


Figure 11 Logical Channel between two Applications

a) the maintenance of the BIU sequentiality;

b) the recognition and the discarding of the duplicate PIU;

c) the recognition of the PIU loss;

d) the syncronisation of the two Applications on the LC.

4.2.2.5 LC Multiplexing/De-multiplexing. The activity of all the LCs is multiplexed on the connection with the Common Network, and the incoming traffic is distributed among the active LCs.

From the functional point of view, the SH can be considered subdivided in four sections (considering as a separate section that provides the error handling), which will be described in detail in the following sections.

4.2.3 Presentation Services

The presentation Services provided to the Applications are:

a) handling of the Flip-Flop discipline of the LC;

b) RU chain control;

c) response type control.

In the Flip-Flop discipline the Sender/Receiver relationship on the half duplex LC is fixed at any time, and can be reverted only on initiative of the Sender. The protocol is based on the status of an indicator (Change Direction), associated to each data transfer request at the Sender side. The rules governing the Presentation Services are very simple and regard the following controls:

a) Chain Control;

b) Change Direction Control;

c) Response type Control.

The most part of the control is carried on at the Sender side, in order not to send data affected by a rule violation. However, some security checks are made also by the Receiver, and where a violation of the rules is recognized, the appropriate exception is generated.

All the information necessary to the Presentation Service control are carried by the BIUs exchanged during the session. The field that contains this information is the Request Header (RH), and this is the first field of each BIU.

4.2.3.1 Request Header (RH). The Request Header (RH), which is attached to each RU as the RU passes through the Presentation Service Control, contains all the indicators for the Presentation Service Protocol. The RH is two bytes in length, and the information carried by it is the following:

a) Action Code Field (ACF)

b) Action Modifier Indicator (AMI)

c) Change Direction Indicator (CDI)

d) Chain Control Field (CCF)

The above control fields are defined as follows:

a) specifies the action to be performed on the information which follows;

b) contains the indicators that modify the action code meaning;

c) this control indicator can be used during a session, by the Sender in order to reverse the LC data flow direction, to assume the Receiver role;

d) this field establishes which element of a chain of RUs this RU is: first, middle, last, or only member of a chain. RUs that are chained together form a unit. Two indicators, "begin chain" and "end chain" provide the chain control information.

4.2.3.2 Chain Control A "Chain" consists of a set of related RUs which are recognized as an entity through the network. A chain has an RU which is the first of the chain (marked "begin chain"), which may be followed by one or more Middle Chain RU(s) (marked "middle"), and ended by an RU that is the last in chain (marked "end chain"). A chain may consist of many RUs, or only one RU; the chain as an entity is the basic unit for the recovery.

In the following scheme the rules for processing chains are given. These rules are based on the assumption that the "begin chain" and "end chain"indicators are checked separately first the "begin chain", then the "end chain", every time following the appropriate rules.

4.2.3.2.1 Chain Control at the Sender side.

- The first RU in chain is marked "begin chain", and puts the Sender into the "in chain" state.

- If a negative response to the current chain is received while in the "in chain" state, the "between chain state" is entered.

- The last RU in a chain is marked "end chain". It returns the Sender into the "between chain" state. Middle RUs in chain do not affect the chain state.

- If in the "between chain" state the Application program tries to send "middle" or "last" in chain RU, an error indication is returned to the Application program and the RU is not sent. The Chain control status remains unchanged.

- If in the "between chain" state a response is received, the chain control status is not changed, and the response is forwarded to the Application program.

- If in the "in chain" state the Application program tries to send a "begin chain" RU, an error indication is returned to the Application program, the chain control state is not changed, and the RU is discarded.

4.2.3.3.2 Chain Control at the Receive side.

- If in the "between chain" state, a "begin chain" RU puts the chain control into the "in chain" state.

- If in the "between chain" state, middle in chain or end chain RUs are discarded, a negative response (when required) with a sense indication of chaining error is given, and the "purging chain" state is entered.

- If in the "in chain" state, middle in chain RUs do not affect the chain control state.

- If in the "in chain" state, and an end chain RU is received, the chain control returns to the "between chain" state.

- If in the "in chain" state, and a sequence error is recognized, the "purging chain" state is entered.

- If in the "purging chain" state, purging is stopped by the receipt of an "end chain". When purging stops, the chain control enters the "between chain" state. The end chain indicator has to be examinated on all chain elements, even those elements which caused the "purging chain" state to be entered.

- If in the "purging chain" state, purging is stopped by the receipt of a "begin chain" RU. When purging stops, the chain control enter the "in chain" state.

- If in the "purging chain" state a "middle in chain" RU is received, the chain control discards the RU, no response is generated, and the state remains unchanged.

- As a general rule, a negative response is generated when a sequence error is recognized, in whatever state the chain control is.

The situation can be summarized in the flowcharts of Figure 12.

4.2.3.3 Change Direction Control. The Change Direction (CD) indicator is used in the Presentation Services to exchange the Send-Receive role between two Applications during a session. Only a request on the normal (Sender to Receiver) flow which is also marked "end chain" or "definite response" may carry a CD. When the sender includes CD in a request, it indicates that the sending Application is prepared to receive, and that the other Application can send. The receiver is required to control the CD. In the following the rules for processing the Change Direction Indicator are given.

- The Sender when receives a SEND request with CD, enters the "exchanging role" state. In this state (Pseudo - Receiver) only RECEIVE requests are accepted.

- If a RU is received without error while in the "exchanging role" state the "receive" state is entered.

- If a RU is received in error while in the "exchanging role" state, the state remains unchanged.

- If an error indication carrying CD not accepted or just negative acknowledging the last request is received while in the "exchanging role" state, the "sender" state is entered, and the pending RECEIVE request (if any) is closed with the "CD not accepted" exception.

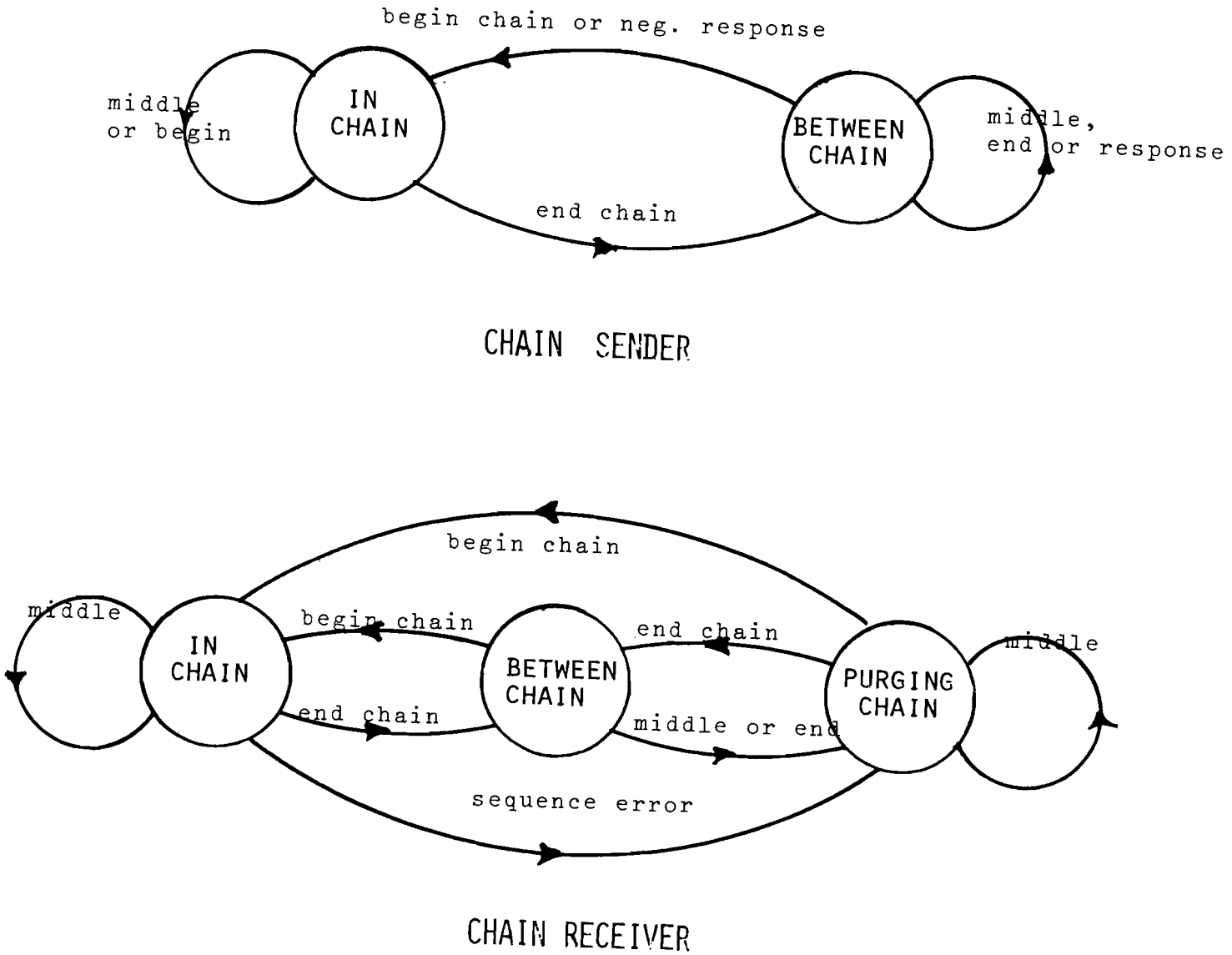


Figure 12 Chain Sender and Chain Receiver

- If the CD indicator is received and recognized, while in the "receiver" state, the "exchanging receive role" is entered. In this state, the mode of the response is forced as "definite".

- If the CD indicator is received together with an exception indication on the request carrying CD, the state is not affected, and a negative response is issued, with a sense code of CD not accepted.

- While in the "exchanging receive role" state, a SEND operation terminates without error, the "send" state is entered.

- While in the "exchanging receive role" state, if the SEND operation terminates with error, the state remains unchanged, and the operation is closed with error to the Application.

To be noted that, due to this mechanism, the Sender, in case of CD, will not be notified until the first RECEIVE operation will be closed. The exception (if any) will be reflected only on the "old" Receive side. This is the only case in which the LC operations are unbalanced.

The above operations are depicted in the flowchart of Figure 13.

4.2.3.4 Response Type Control. The "Form of response requested" field is used to establish the response action. There are three options with regard to requesting responses: the Sender may request no responses (No Response), a response only if an error occurs (Exception Response), or a response whether or not an error occurs (Definite Response). As far as the data flow control is concerned, when "No Response" or "Exception Response" is choosen, the Sender can continously send requests. Instead, when the "Definite Response" is choosen, the Sender must wait for the response to a request before sending another one. At the Receiver side, the "Form of Response Requested" field is used only to establish the response action, but no data flow control is involved. These rules are affected by the existence of RU chains: each "chain" is treated by the Sender and the Receiver as a single request. This high level request can be sent in one of the three possible modes: No-Response, Exception Response, Definite Response. In order to handle this high level request, each RU in the chain is sent in the appropriate mode:

- for "No Response" chain, each element of the chain is sent in No Response mode;

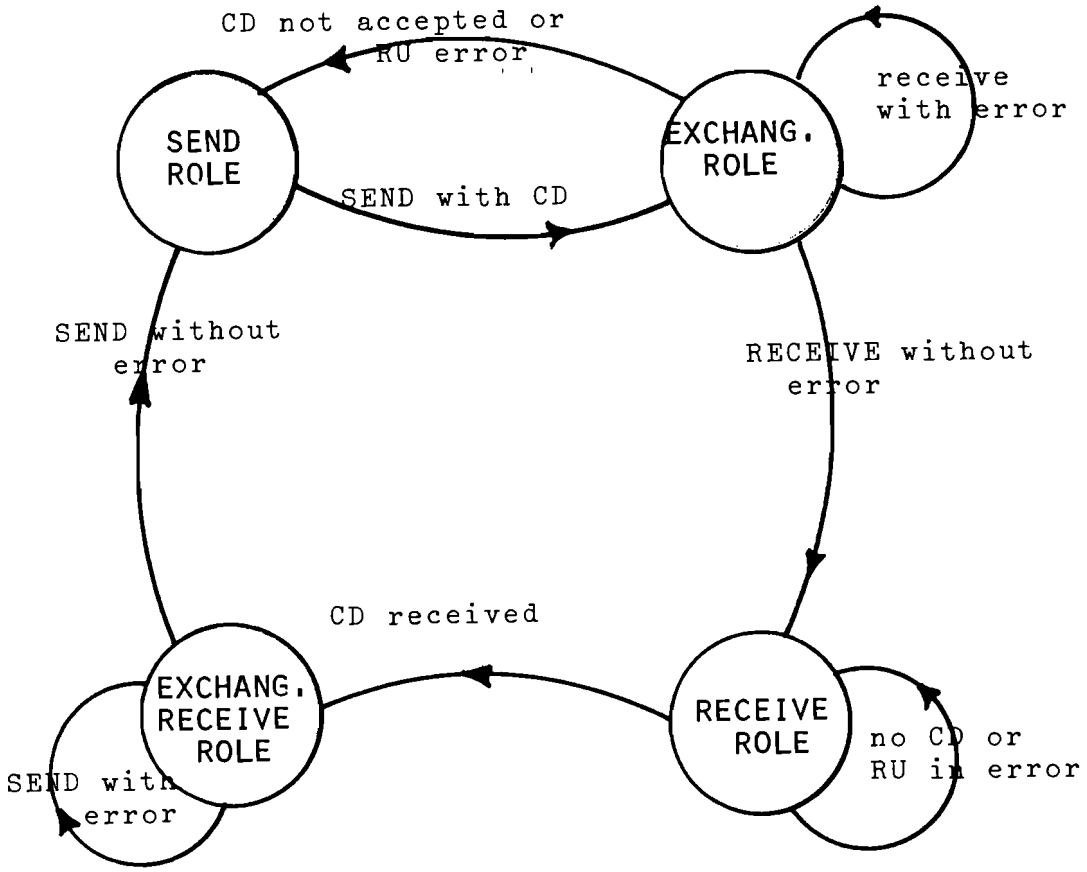


Figure 13 Change Direction Control

- for "Exception Response" chain, each element of the chain is sent in Exception Response mode;

- for "Definite Response" chain, the last element of the chain is sent in Definite Response mode, the others are sent in Exception Response mode: this to have only one response for the chain, considered as an entity.

Only these types of chains are allowed to be sent. The Sender of a chain has the responsibility of the appropriate setting of the "Form of Response Requested" field for each element of the chain. The Receiver of a chain is required, as response type control, to examine the "Form of Response Requested" field only on the last element of the chain, unless an error occurs earlier in the chain.

For security purpose, the "Form of Response Requested" field is associated to each BIU or BIU segment, and for this reason, it is carried by the Transmission Header (TH).

4.2.4 Data Length Adapter

The length of the data unit (RU) which can be exchanged on the Logical Channel is limited, but that limitation is unrelated to the data unit length limitation, which is a characteristic of the Common Network used, generally more restrictive.

Pratically, there is the need to adapt the BIU length (RH+RU) to the Common Network characteristics. This necessity can be satisfied having a mechanism to segment the outgoing BIUs (when necessary), and to reassemble the incoming BIU segments. Each outgoing BIU is segmented (when necessary) and prefixed with addressability, identification and sequential order information (Transmission Header, TH). A TH plus a BIU or a BIU segment is called a Path Information Unit (PIU), that is the data unit accepted by the Common Network. The TH contains all the information to identify the BIU and the BIU segment inside the BIU. The PIUs are passed to the Common Network in order to be forwarded to their destination. Each PIU pertaining to a segmented BIU is sent separately and the next in sequence segment is not sent until the previuos one has not been positively handled by the local transmission subsystem (Common Network). This rule has been introduced in order to multiplex the local LC outgoing traffic at the PIU level, and to control the BIU segment sequentiality at the first step of the transmission path, reducing the probability of out of sequence arrivals at the destination (the Common Network does not maintain the PIU sequentiality).

The identification of each PIU is accomplished by means of three fields of the Transmission Header (TH), and precisely:

- the PIU number, which is an incremental and cyclic counter, identifying the order number of the PIU within the session;
- the BIU number, which is an incremental and cyclic counter, identifying the order number of the BIU within the session;
- the Segment Mapping Field, which is a two indicator field that identifies the relative position of the segment within the BIU: first, middle, last or only BIU segment.

All the PIUs marked "first" or "middle" BIU segment are fixed in length (the length being the maximum length allowed by the Common Network); the PIUs marked "last" or "only" BIU

segment are variable in length up to the maximum allowed.

The incoming PIUs, filtered by the Data Flow Control (DFC) section, are to be reassembled in order to reconstruct the original BIU. Because of the possible out-of-sequence arrival of the BIU segments (PIU), the BIU data integrity has to be controlled during the reconstruction. When the reconstruction cannot be completed, because the DFC has recognized a PIU loss, the BIU is declared in exception and discarded.

The segment identification is organized in such a way that it is possible to put the single segment in the correct position inside the reconstruction buffer whichever the arrival order is.

4.2.4.1 Transmission Header. The Transmission Header (TH) is that part of a PIU which provides addressability, identification and sequential order of the BIU or BIU segment carried by the PIU as its text part. The network addresses are carried as Destination Address Field (DAF) and Origin Address Field (OAF) by the TH. The identification and sequential order are carried by three fields, as mentioned before:

- the BIU sequence number (BSN);
- the PIU sequence number (PSN);
- the Segment Mapping Field (SMF).

The Data Count Field (DCF) contains a binary count of the text (BIU or BIU segment) carried by the PIU. The scheme of the TH is depicted in Figure 14.

where

PTY Priority (1 byte): contains the priority with which this PIU will be handled within the Common Network. In the first implementation this field is unused.

DAF Destination Address Field (3 bytes): contains the network address to which the associated PIU has to be forwarded.

OAF Originator Address Field (3 bytes): contains the network address of the originator of this PIU.

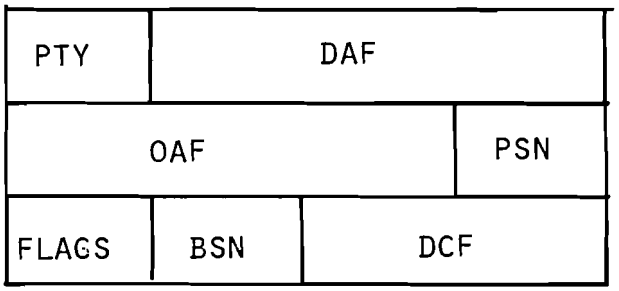


Figure 14 TH structure

PSN PIU Sequence Number (1 byte): contains the sequence number of the PIU. This incremental number is cyclic, and represents the order number of the PIU within the session.

FLG Flags (1 byte):
SMF (Segment Mapping Field) : B'00000011' this field represents the relative position of this PIU inside the BIU to which it pertains. Two indicators, "First" (B'00000001'), and "Last" (B'00000010), give the possibility to define four states: first segment, middle segment, last segment and only segment.
RRC (Requested Response Control) : B'01000000' It defines the form of the requested response. The possible types are: No response, Exception response (B'00100000') and Definite response (B'01000000').
LOC (SH Internal Message): The internal message is a protocol message: this is signalled by this flag (B'10000000').

BSN BIU Sequence Number (1 byte) contains the sequence number of the BIU to which this PIU pertains. This number is cyclic and it is never reset. The Break count is merged into this field.

DCF Data Count Field (2 bytes) contains a binary count (in bytes) of the text carried by the PIU.

4.2.5 Data Flow Control

The Data Flow Control (DFC) section of the Session Handler has to provide the following functions for each active logical channel:

1 maintenance of the BIU sequentiality;

2) recognition and discarding of duplicate PIUs;

3) recognition of PIU loss;

4) syncronisation between the two Applications on the Logical Channel during the session;

and moreover it has to:

5) avoid the buffer congestion;

6) prevent the deadlocks.

The DFC to DFC protocol that performs the previously stated characteristics of the data flow is described in what follows.

Basically, the DFC to DFC protocol uses the well-known window strategy, and acts on the PIU flow. Supposing that the PIU sequence number field in the TH allows sequence numbers to range from Ø to n-1, the Sender will not transmit more than W PIUs, without receiving an acknowledgment, the W being the window. Of course, W must be less than n.

The basic rules for the Sender and the Receiver are described in the following. The structure of the window is sketched in Figure 15.

SENDER: Let M be the sequence number associated with the window left edge.

a) The Sender transmits only PIUs whose sequence number lies between M and up to M+W-1. If the rigth edge of the window is reached without receiving any acknowledgement, the activity is holded waiting for a "receiver's acknowledge status" (from now on briefly "status").

b) On receipt of a "status" consisting of the Receiver's current window left edge and window width, the Sender's left window edge is advanced over the acknowledged (positive or negative) PIUs, and the rigth window edge is adjusted

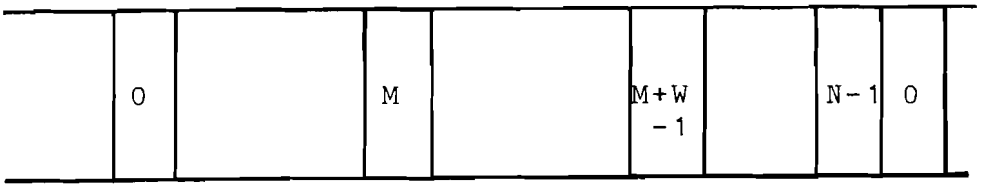


Figure 15 The Dynamic Window

consistently. The "status" may be received at any time. No recovery is provided for the negative acknowledged PIUs.

RECEIVER: Let M be the sequence number associated with the left edge of the current Receiver's window. The rigth edge is defined as M+W-1, where W is the current window width.

a) Every PIU received with a sequence number inside the "dynamic window" is accepted or discarded if already received. As "dynamic window", the Receiver considers the window defined by the last "status" sent.

b) PIUs arriving with a sequence number outside the "dynamic window" edges are discarded as duplicates.

c) The "status" is sent to the Sender every "half window" that is when the PIU with sequence number M+W/2-1 or over and M+W-1 or over have been received. The "status" contains the "dynamic window" left edge and width. The width is calculated according to the memory availability. This information is dynamically updated on PIU receipt, considering that part of the window which contains only the PIUs received or declared loss.

d) The receipt of a PIU with a sequence number over the rigth edge of the "current window" is considered as an indirect acknowledgment of the "status" sent, so that the current window is updated following the "status" information, and the acknowledged "status" is purged.

4.2.5.1 How the protocol works. Before to precise the DFC protocol, some general considerations are to be made. No mechanism is provided by the Common Network to avoid a buffer congestion at the Session Handler level. SH has to protect itself against the possible buffer congestion induced either by out-of-sequence PIU arrival and by a poor Application synchronisation.

In both the cases, a certain amount of storage is kept busy waiting for BIU completion or for a RECEIVE request from the Application. Moreover, there is the possibility of PIU loss during its journey through the Common Network, that means an infinite out-of-sequence for the last PIU, and consequently the SH, on the related Logical Channel, may be kept waiting indefinitely for BIU completion or to provide the correct BIU sequentiality.

In the hypothesis of no PIU loss, the DFC protocol could avoid buffer congestion, controlling the window width (W), considering the buffer availability and the Application synchronisation. The out-of-sequence causes delay in BIU delivering, and in window advancing, pratically reducing the Logical Channel bandwidth.

The same happens in case of poor synchronisation between the two Applications. Clearly, no precautions can be taken by the DFC against a bad synchronisation between the Applications, because this is out of the DCF control. The only possibility to improve the Logical Channel bandwidth, assuming at the best the buffer availability is to reject, as an error, long out-of-sequence, where with long out-of-sequence is intended an out-of-sequence with very little probability to happen, practically out of the range of the most part of the observed out-of-sequence frequencies. In this case,if the Logical Channel bandwidth is improved, the reliability is reduced, because a PIU is declared in error without an objective prove of the error itself. When the possibility of PIU losting is considered, deadlock situations are to be resolved. Such situations are due both to Application PIU loss and to "status" PIU loss. The DFC protocol, as it has been described, is unable to avoid or recover these deadlock situations, so that it has to be completed in order to consider also the possibility of information losting.

The following hypothesis, based on the Common Network characteristics, can be made: a PIU loss is always connected with a reductive change in the Common Network topology; in other words, only when a Node of the Common Network passes from the active to the inactive state (due to a software or a hardware failure), there is the possibility of a PIU loss.

Because in RPCNET the logical Network Control Point (the Network Services Manager) becomes acquainted with every change in the Common Network active topology, the DFC protocol mechanism to avoid deadlock situations may be triggered by the knowledge of a reductive reconfiguration in the Common Network.

To summarize, there is a number (M) that represents the maximum out-of-sequence acceptable by the DFC; when this number is exceeded by a long out-of-sequence, the related PIU is considered lost, and the window is updated consistently. In case of a reductive reconfiguration of the Common Network, DFC provides a recovery mechanism to avoid deadlock situations, that is, every Logical Channel which at the moment is in the "waiting for status" state (the only state in which there is a deadlock possibility), sends a "Information for status" to its counterpart, to solicit the "status" and to resynchronize the session. At the Receiver side of the concerned Logical Channel(s), the receipt of an "Information for status" freezes the window state considering in error (lost) all the PIUs not arrived at the moment, and whose number lies between the left window edge and the number reported in the "Information for status" (last PIU sent). In any case, no deadlock is possible, because either all the missing PIUs arrive soon or later, or a reductive reconfiguration is signalled, starting the recovery mechanism (in case of Definite Response requested, or end of window reached), or new activity on the Logical Channel allows to clarify the situation.

In order to use the DFC protocol, three parameters are to be defined: the number (N) of the PIU numbering cycle, the width (W) of the window, and the number (M) of the maximum end-of-sequence accepted.

(N) is defined by the maximum bynary number definable in the PIU Sequence Number Field (SNF) of the TH. The PIU SNF is one byte width, that means a numbering cycle of 256 (from Ø to 255). As a matter of the fact, the PIU identification cycle, as far as the duplicate recognition is concerned, may be considered higher, because beside the PIU SNF there is also the BIU SNF to complete the PIU identification. These two sequence numbers (unrelated each other) may rise the cycle up to 65.536.

(W) Its value is mainly related to the receiving SH buffer availability, because all the PIUs inside the window not pertaining to the BIU presently under reassembling or waiting for a RECEIVE request, have to be maintained in memory.

Clearly, (W) is not univocally and statically definable, because its value depends on many factors: on the dynamic SH buffer availability, on the type of the Logical Channel activity, and on the Common Network crossing delay (this to avoid, when possible, stops on the sending activity due to the fact that the end-of-the-window is reached before receiving the half-window "status"). Anyway a relation among (N) and (W) can be fixed: $W < N/2$ to avoid that the arrive of an old PIU may appear as a new transmission (N/2 should be a convenient number). Just to make a guess (to be revised after a test period), in the first implementation it was adopted: $W = 16$ (as first value at the session opening time: this value will be recomputed dynamically during the session).

(M) Its possible value is a therothical mystery; there has been a first attempt to evaluate this number using the simulation approach. The model used is referred to a static Common Network (no reconfiguration considered) using the "optimal path" as routing technique. The result is a very little probability of out-of-sequence and a maximum observed out-of-sequence of the same order of the number of the crossed nodes. This information is not sufficient to establish a mean value for (M); a relationship was established between (M) and (W): $M < W/2$ to avoid that, using the long out-of-sequence technique, the uncertainity on the PIU arrival shall be over half window. In the first implementation, a value was assumed for (M): $M = 4$

4.2.5.2 Complete DFC Protocol. The complete DFC protocol can be described using the state diagram technique:

SENDER control

a) When in IDLE state, and a request to send a PIU is received, the SENDING PIU state is entered.

b) When in IDLE state, a receipt of a "status" puts the SENDER control in UPDATING WINDOW state.

c) When in SENDING PIU state the PIU sent acknowledged from the Common Network of a PIU whose sequence number is inside the window, returns the SENDER control in the IDLE state.

d) When in SENDING PIU state the PIU sent acknowledged from the Common Network of a PIU whose sequence number is the last permitted by the current window or of a PIU that is the last segment of a BIU requesting Definite Response, puts the SENDER control in WAITING FOR STATUS state.

e) When in WAITING FOR STATUS state the receipt of a "status" puts the SENDER control in UPDATING WINDOW state.

f) When in WAITING FOR STATUS state, the receipt of a "status" which does not reset the Definite Response request. does not change the SENDER control state.

g) When in UPDATING WINDOW state, the window updated condition puts the SENDER control in the IDLE state, or, if the updated window width is Ø, the control returns to the WAITING FOR STATUS state

h) When in WAITING FOR STATUS state, the signalling of a reductive reconfiguration of the Common Network puts the SENDER control in SENDING REQUEST FOR STATUS state.

i) When in SENDING REQUEST FOR STATUS state. the request sent condition returns the SENDER control to the WAITING FOR STATUS state.

l) When entering the WAITING FOR STATUS state, the existence of a reconfiguration signal inside the actual window puts the SENDER control in SENDING INFORMATION FOR STATUS state, and clears the situation, then the Sender control returns to the IDLE state.

RECEIVER control

a) When in IDLE state, the receipt of a request for "status" puts the RECEIVER control into the CLEAR WINDOW state.

b) When in IDLE state, the arrival of a PIU puts the RECEIVER control in the CHECK SEQUENCE NUMBER state.

c) When in CHECK SEQUENCE NUMBER state, a PIU with a sequence number out of the "dynamic window" causes the PIU to be discarded, and the RECEIVER control returns into the IDLE state.

d) When in CHECK SEQUENCE NUMBER state, a PIU with a sequence number inside the "dynamic window" (and not yet received) causes the PIU to be accepted, and the RECEIVER control enters into the UPDATE WINDOW state.

e) When in UPDATE WINDOW state, the receipt of a PIU whose sequence number is at the SENDER "status" trigger point causes the RECEIVER control to enter the SENDING STATUS state.

f) When in UPDATING WINDOW state, the receipt of a generic PIU causes, after the window updating, the RECEIVER control to enter the IDLE state.

g) When in SENDING STATUS state, the dynamic window is updated, and the new status message is sent. The RECEIVER control returns into the IDLE state.

h) When in UPDATING WINDOW state, if the left edge of the updated dynamic window falls into a PIU which is the last segment of a BIU requesting Definite Response, the RECEIVER control enters the SENDING STATUS state.

i) When in UPDATING WINDOW state, the receipt of an "information for status" updates the window status, using the "information for status" information, and if status point has been reached, the control passes to the SENDING STATUS state, otherwise the control passes to the IDLE state.

l) When in CLEAR WINDOW state, all not received PIUs, until the last PIU sent (in the "status" message), are declared lost. The control is passed to the SENDING STATUS state.

The situation is depicted in the flowcharts of Figure 16 and Figure 17.

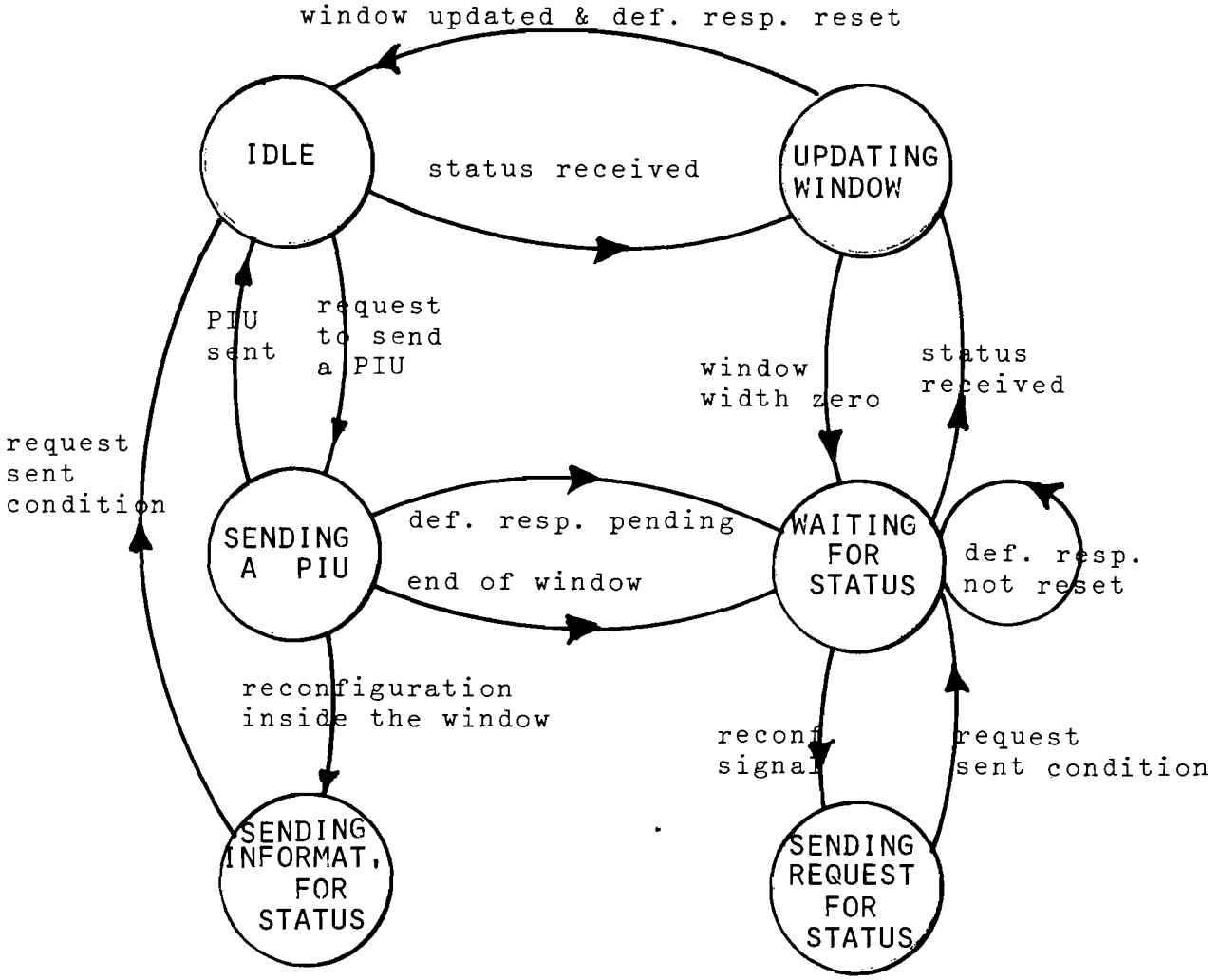


Figure 16 Data Flow Control: Sender side

4.2.6 Session Exception Handler

Due to the layered structure of the Session Handler, the exceptions, like the data flow, have to cross all the layers, in order to collect the appropriate error condition

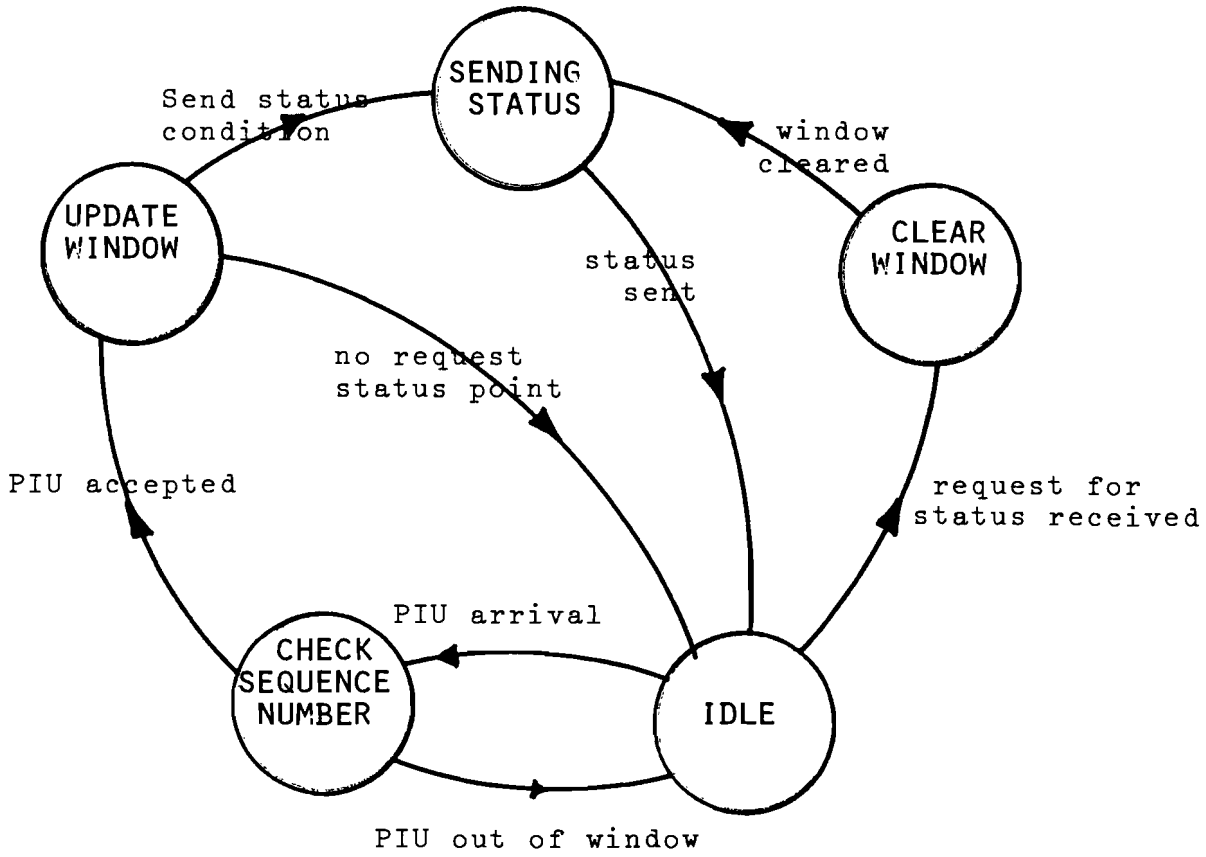


Figure 17 Data Flow Control: Receiver side

(to be reflected to the Application), and to put the single layer functions in their appropriate states. Moreover, as in a session there are two Applications linked together, both have to be notified of eventual exceptions, wherever the exception has been recognized. Practically, the only error condition is the PIU loss, recognized by the Data Flow

Control layer. The PIU loss exception induces the exception for the BIU to which this PIU pertains: this is recognized by the Session Exception Handler (SEH), which provides:

a) the insertion of this exception in the "status" message in order to forward the "BIU in exception" information to the session counterpart;

b) the signalling of the exception to the Data Length Adapter (DLA) layer and to the Presentation Service Handler (PSH) layer, when the BIU in exception becomes (as sequence number) the "actual" BIU, that means when this BIU is explicitly requested by the Application via a RECEIVE operation;

c) the reflection of the exception together with the eventually collected error conditions, to the Application (closing in exception the RECEIVE operation).

Analogously, when the SEH is informed of a BIU exception via the "status" message mechanism, it has to provide:

a) the signalling of the exception to the DLA and PSH layers;

b) the reflection of the exception together with the eventually collected error conditions to the Application. This reflection may be synchronous or asynchronous, depending on the way of proceeding chosen by the Application for its requests (synchronous means Definite Response, asynchronous means Exception Response).

The exceptions recognized directly by the PSH on requests that are semantically incorrect, do not proceed through the other layers, but are immediately reflected to the Application originating the request.

The possible error condition are:

Basic error:

- BIU data integrity failure (the loss of one (or more) PIU(s) pertaining to the BIU causes this failure, and the BIU is discarded).

Induced error:

- BIU chaining failure (the loss of a BIU that is chained invalidates all the chain of BIUs).

- Change Direction not accepted (the failure of a BIU carrying the Change Direction request causes the rejection of the Change Direction, and the return of the control to the original Sender).

Warning Information (not considered error):

- BIU data overflow (the received BIU is longer than the buffer allocated by the RECEIVE operation; this exception is presented only to the Application at the RECEIVE closing time).

4.2.7 A final remark

In the actual implementation, the "send status" triggering event is the memory availability. In fact, if there is no memory available when a status has to be sent, the preparation of the status message is delayed until when there is memory availability. This condition was established to prevent the sending of a status with window width equal to zero, and then another status when the window width may be enlarged.

4.3 The Network Services Manager

The Network Services Manager (NSM) is an Interface Functions component which provides for the control of the Logical Units (LUs) and the Logical Channel Terminations (LCTs), and supplies a set of services to the LUs and LCTs which it controls.

The NSM maintains knowledge of the states of all LUs, LCTs, and of the sessions which the LCTs are participating to. The NSM itself is a system LCT, and it maintains permanent sessions with all the other NSMs, sharing the Interface Function Layer control. The NSM is in permanent contact with the CNM of the node to which it belongs, in order to have an up-to-date knowledge of the real network configuration.

The set of services used to control the Logical Network are called Network Services. The NSM may send Network Services requests to other NSMs in order to perform the functions requested. Every request for service to the NSM by the Application has to be made through an LU. In case where the service is requested by another NSM, the NSM is responsible for its execution and subsequent notification of the originator.

Network services are divided into categories according to the functions being performed. The categories defined are:

a) Measurement services

b) Network Operator services

c) Session services

d) Mailing services

e) Inquiry services.

The Network Services functions may be invoked only by a request containing coded information. The codes are detailed in Appendix C.

The NSM, as system LCT, mantains multiple sessions with all the other NSMs in the Network. Each Multiple session is handled by the Multiple Session Handler. The data unit (BIU) exchanged between NSMs are monosegment, i.e. the maximum size of a data unit is equal to the maximum size of the PIU in the Common Network. The Logical Channel related to each

one of the session with the other NSMs can be considered as a full-duplex connection with the following characteristics:

a) Error free connection.

b) Data units limited in size (same limit as the Common Network).

c) Data integrity mantained.

d) Data unit sequentiality not mantained.

e) Data unit duplication impossible.

In a NSM, a number of functional units can be recognized; they are:

a) Logical Unit Handler (LUH): Each Logical Unit represent a port on the Communication System through which an Application can access the Network environment. The LUH is that part of NSM which handles and controls all the LUs activity.

b) Logical Network Control Point (LCP): LCP provides for the control of the configuration of the Logical Network, that is, it knows at any moment the status of each Network Station, as far as the connectivity is concerned (reachable or unreachable). Every change in the Network Station is notified to LCP by the Physical Network Control Point (CNM). Moreover, LCP provides for the control of the validity of the active sessions, every time a reductive reconfiguration is signalled. LCP routes this information to the Session Handler (SH) in order to trigger the session recovery mechanism.

c) Multiple Session Handler (MSH): The NSM maintains permanent sessions with all the other NSMs in the network. All the sessions are to be maintained and controlled, and moreover a recovery has to be provided for the BIU signalled in exception. These functions are performed by the MSH.

d) Network Services Switcher (NSS): All the request for service are presented to the NSS which in turn switches the request to the appropriate component of the Services Processor. NSS acts also as an interface between the Services Processor and the Multiple Session Handler.

e) Services Processor: The Services Processor is composed of as many modules as the Service functions are:

1 The Session Service provides the initiation and termination of the Sessions between the LUs. A LU may request a session either as a primary LU, via the BIND request, or as a secondary LU, via the INVITE request. Both LUs in a Session may request the termination of the Session via an UNBIND request.

2) The Network Operator Service provides the local operator with the ability of monitoring the Network activity.

3) Measurement Service provides for measurement of the utilisation of the Network resources in the domain of the control of the NSM. Measurement results are collected for further statistical and/or accounting purposes.

4) Inquiry service provides the possibility to inquire about the structure and the resource status of the Network environment. A LU may request information about the Network environment via an INQUIRE request.

5) Mailing Service provides a LU with the possibility to send and to receive messages (letters) to and from another LU in the Network. A LU may use the MAIL request to send a message (letter) or to set up a mailbox in order to be able to receive messages (letters).

4.3.1 The Logical Unit Handler

The Logical Unit (LU) may be considered the port on the Communication System through which an Application can access the Network environment. The activity of the LU is composed of the coded (and formatted) requests for the services available in the Network environment.

An Internal Address (a binary number in the range from 0 to 255) is assigned to each LU for internal addressing purposes between the Application and its Communication System, namely the NSM. When a Session is established, a Network Address is associated to the LU together with a Logical Channel Termination (LCT). The LU, through its LCT, becomes directly addressable in the Network and its control passes to the Session Handler (SH), which handles every in-

session request. When the Session is closed, the LCT is released, and the LU returns under the direct NSM control.

The maximum number of LUs opened at the same time (theoretically 256) is a local parameter, and it is charge of the LUH to respect this limit. A LU is assigned to an Application on direct Application request, and it is released on Application request. No more than one request at a time can be present on the same LU. The LUH has to control the flow of the requests on the single LU, and to enforce that limit. When a request is completed, a completion code with the eventually requested data is returned to the Application. The LUH is now ready to receive the next request. Asynchronous activity may be possible on a LU throughout the LU like exception signalling or asyncronous data (mail) arrival.

The LUH is responsible for the execution of the OPENLU and CLOSELU requests issued by the Application (actually, in the VM implementation, the OPENLU request is carried on by the NAC, which will be described in the next section, while in the OS implementation it is carried on by the NSM, but this is absolutely transparent to the user, that is the Application: it depends on the operating system under which RPCNET is implemented).

Moreover, the LUH has to handle the, in case, abnormal end of an Application signalled by the system (operating system, or anything similar), informing the CNS of the forced closing of the associated LUs. The CLOSELU request is the only request which can be issued by an Application whichever the LU status shall be (free or busy). On CLOSELU request the NSM has to cancel eventually pending operations or has to close an opened session (at the moment). The OPENLU request is the only request which does not refer to an existing LU

4 3.2 The Logical Network Control Point

The Logical Network Control Point (LCP) is the active control point of the Interface Layer as far as the local Network activity is concerned

The LCP is in permanent contact with its Node Common Network Manager (CNM), in order to be updated on the Network connectivity. Every change in the Network connectivity will be handled by the LCP, checking the local Session validity and starting tne, in case, recovery mechanism.

When a Network station becomes unreacheable, due to a Host, a Node or a Link failure, all the (local) sessions

opened at a moment with the now unrecheable Network station are forced to close, and all their pending activity is cancelled. The Applications are informed of the failure, the LCTs are released, and the corresponding LUs are reset to their basic state.

When a possible Node failure (Node unreachability) is recognized, the Session Handler is alerted, in order to start the recovery mechanism for the perhaps blocked sessions (see SH Data Flow Control).

The LCP may be informed by the LUH of System (Application failure), or Application immediate closedown of the active sessions (LCTs). In this case, the LCP forces UNBIND request for the session and releases its LCT while the corresponding LU will be released by the LUH.

4.3.3 The Multiple Session Handler

The NSM provides permanent sessions with all the reacheable NSMs in the Network. The NSM, as system addressable unit, has an unique address, the zero address, in every Address Space (Network Station) of the Network.

All the NSM to NSM sessions are implicitly established at the starting time, so there is no need to explicitly establish the single session. Every NSM to NSM session is identified, as every session does, by the couple of Network addresses identifying the two session partners.

During a session the two NSMs can be considered connected through a Logical Channel. This Logical Channel can be viewed as a full-duplex connection with the following characteristics:

a) error-free connection

b) data unit limited in size (the same limit as the Common Network)

c) data integrity maintained

d) data unit sequentiality not mantained

e) data unit duplication impossible.

In order to build the previous characteristics using the Common Network (see SH Data Flow Control), the NSM has to use an internal protocol. The Multiple Session Handler (MSH) has to provide the following functions for each active NSM to NSM connection:

a) recognition and discarding of the duplicate PIUs;

b) recognition and recovery of PIU loss;

and moreover it has to:

c) avoid the buffer congestion;

d) prevent deadlocks.

The protocol used to build the previous stated functions is described in what follows.

Basically the protocol uses the window strategy and acts on the PIU flow (no multisegment data unit between NSMs are allowed). Supposing that the PIU sequence number field in the TH allows sequence number to range from (Ø) to (n-1), the Sender will not transmit more than (w) PIUs without receiving an acknowledgment, the (w) being the window. Clearly, (w) must be less than (n), and equal for all the NSMs in the network.

Each data unit exchanged between NSMs cannot be multisegment, that is, the restriction in size valid in the Common Network is to be respected by the NSM to NSM data traffic. No control on the data unit sequentiality is required, because each data unit is independent from all the others, and no data limit numbering is required. The data units received and accepted are routed to the Network Services Switcher, which in turn will provide the switching to the appropriate Service Processor module. In the protocol description the term PIU will be used to indicate the data unit exchanged between NSMs.

The basic rules for the Sender and the Receiver are as follows:

SENDER:

a) the Sender transmits only PIUs whose sequence numbers lies between M and up to M+W-1 (where M is the window left edge, and W is the window width). If the rigth edge of the window is reached without receiving any acknowledgement, the activity is suspended waiting for a Receiver's acknowledge status (briefly "status").

b) On receipt of a "status", consisting of the Receiver's current window left edge, the Sender's window left edge is advanced over the acknowledged PIUs, and the window right edge is adjusted consistently (the window width is a fixed number). The acknowledged PIUs may be freed.

c) If a reductive reconfiguration is signalled by the Common Network, a recovery mechanism is started. All the not acknowledged PIUs are resent

d) Every sent PIU carries as first information the number of the last received in sequence PIU. This number may be used at the other side to free all the pending PIUs whose number is less than or equal to the acknowledged number.

RECEIVER:

a) Every PIU received with a sequence number inside the window and not yet arrived (duplicate) is accepted.

b) PIUs arriving with a sequence number outside the window edge are discarded as duplicates.

c) The "status" is sent to the counterpart every "half window", that is, when the relative "half window" table has been completely filled.

d) When the "status" is sent, the window is consistently updated (the window width is fixed and, of course, equal for all the NSMs in the Network).

e) If a reductive reconfiguration is signalled by the Common Network Manager, and the window is empty (no PIU with a sequence number inside the window has been received) the last "status" is resent.

We will describe the complete NSM to NSM protocol later; now, let us consider an aspect of the Common Network structure.

As it can be seen in Chapter 3, it can happen that a node fails (hardware or software crash) without notifying the partners. In fact, if the node restarts before the timeout on the lines, the CNM on the opposite side is not notified of the partner's fall, so that all continues as

before. On the other hand, at the "crashed" side, all the sessions are closed in error, while the session partners are not notified of the fact. It can happen, depending on the Application protocol, that deadlock situations arise. If we realize that an Application can be (and normally will be) a system program, and that a deadlock situation means also a loss of memory (all the tables must be preserved), it is obvious that a special recovery for this situation must be developed. In RPCNET, every NSM PIU brings a number (modulo 255) called NSM Restart Number. At the arrival of any PIU, the NSM Restart Number is compared with the local one, and if they are not equal, all the LU with that partner are closed, and the SH is alerted, to shut all the sessions. The NSM Restart Number is then set according to the received one. It must be clear that, in case of fault, the last NSM Restart Number must be saved (as a suggestion, on disk).

Now, we can detail the NSM to NSM protocol.

SENDER

a) When in IDLE state, and a request to send a PIU is received the SENDING state is entered.

b) When in IDLE state, the receipt of a "status" puts the Sender control into the UPDATING WINDOW state.

c) When in IDLE state, the signalling of a reductive reconfiguration puts the Sender control into the RESENDING state.

d) When in SENDING state, a PIU sent acknowledgement from the Common Network of a PIU whose sequence number is inside the window returns the Sender control to the IDLE state

e) When in SENDING state, a PIU sent acknowledgement from the Common Network of a PIU whose sequence number is the last permitted by the current window, puts the Sender control into the WAITING FOR STATUS state.

f) When in WAITING FOR STATUS state, the receipt of a "status" puts the Sender control in the UPDATING WINDOW state.

g) When in UPDATING WINDOW state, the window updated condition returns the Sender control to the IDLE state.

h) When in WAITING FOR STATUS state, the signalling of a reductive reconfiguration puts the Sender control into the RESENDING state.

i) When in IDLE state, the signalling of a reductive reconfiguration puts the Sender control into the RESENDING state.

l) When in RESENDING state, the recovery complete condition puts the Sender control into the WAITING FOR STATUS state, or IDLE state, depending on the control arrival.

RECEIVER:

a) When in IDLE state, the arrival of a PIU puts the Receiver control into the CHECK SEQUENCE NUMBER state.

b) When in IDLE state, the signalling of a reductive reconfiguration puts the Receiver control into the RESEND PENDING MESSAGE state.

c) When in CHECK SEQUENCE NUMBER state, an out-of-window condition or an already arrived condition returns the control to the IDLE state.

d) When in CHECK SEQUENCE NUMBER state, a PIU accepted condition puts the Receiver control into the CHECK WINDOW state.

e) When in CHECK WINDOW state, and the sending status condition has been revealed, the Receiver control is put into the SENDING STATUS state.

f) When in CHECK WINDOW state, and the sending status condition has not been reached, the Receiver control is returned to the IDLE state.

g) When in SENDING STATUS state, the status sent condition together with the window updated condition returns the Receiver control to the IDLE state.

h) When in RESEND PENDING MESSAGE state, the window empty condition returns the Receiver control to the SENDING STATUS state.

The situation is depicted in the flowcharts of Figure 18 and Figure 19.

4.3.4 The Network Services Switcher

All the requests for service are presented to the NSS, which in turn forwards the requests to the appropriate component of the Service processor. The requests may arrive both from the Logical Unit Handler that is, from the local NSM activity, and from the remote NSMs activity. Each request for service has to be presented appropriately formatted, because unformatted requests are not allowed. The Network Services are divided in categories according to the functions being performed. The defined categories are (at now):

1 Session Services

2) Mailing Services

3) Inquiry Services

4) Network Operator Services

5) Measurement Services

4.3.4.1 Session Services. The Session Service is in charge of initiating and terminating sessions between LUs. A LU may initiate a session either as a primary LU, via the BIND request, or as a secondary LU, via the INVITE request.

When the session is established the LU becomes a network addressable unit, being the termination of the established logical channel (LCT). Both LUs in a session may request the termination of the session itself via an UNBIND request. The LCT is released and the LU returns under the direct NSM (LUH) control. The session is always established between a primary LU (session request via a BIND) and a secondary one (session request via an INVITE).

In order to set up the session, there is an exchange of messages between the NSMs (Session Services), controlling the involved LUs. This exchange of messages has to observe a well defined protocol. The basic rules of the Session Services Protocol are as follows:

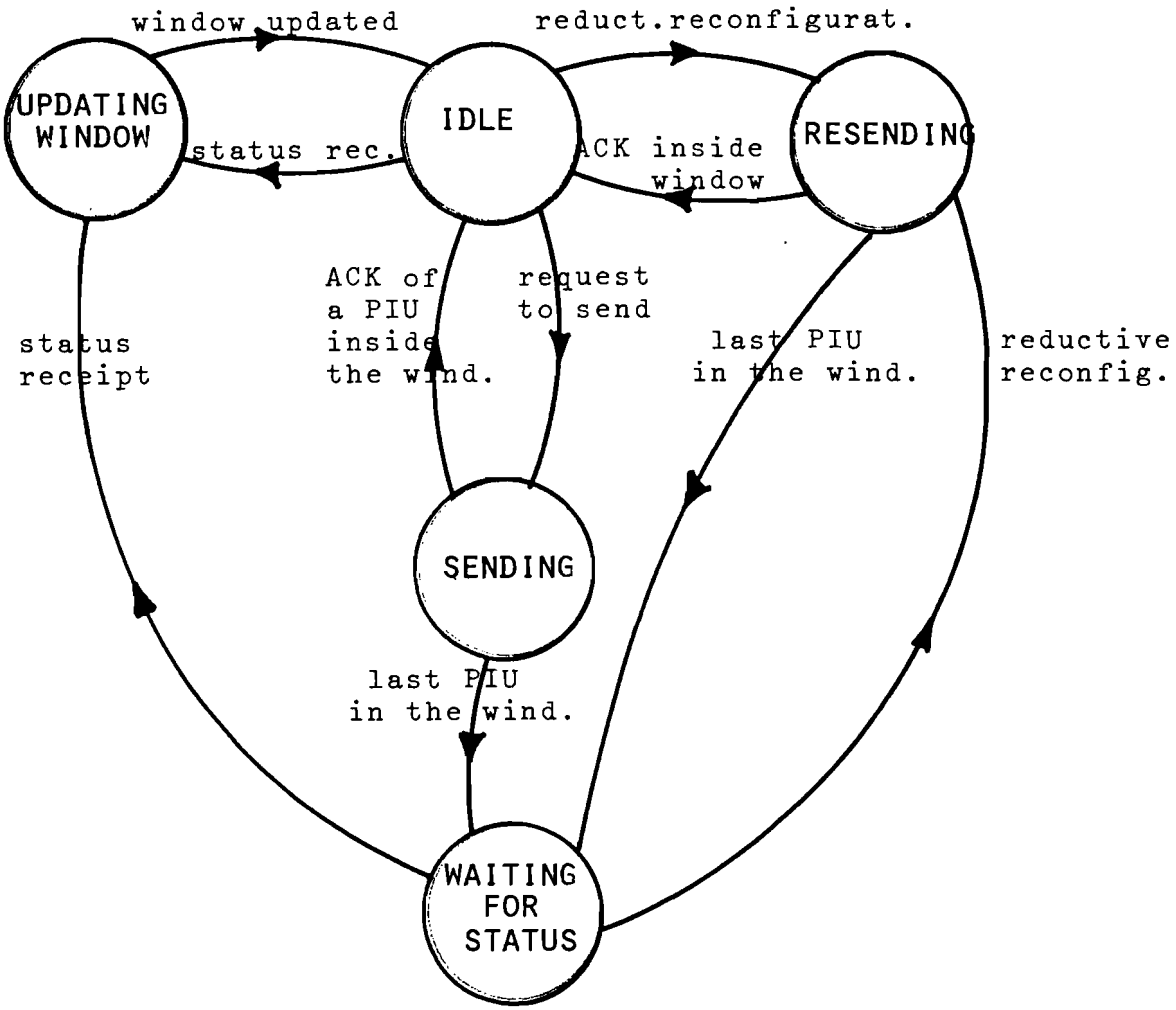


Figure 18 NSM Sender

a) the INVITE request puts the LU in "INVITE" state. The LU is identified by the name of the Application to which it pertains, and it is protected by a password; a network address is assigned, and a LCT is arranged.

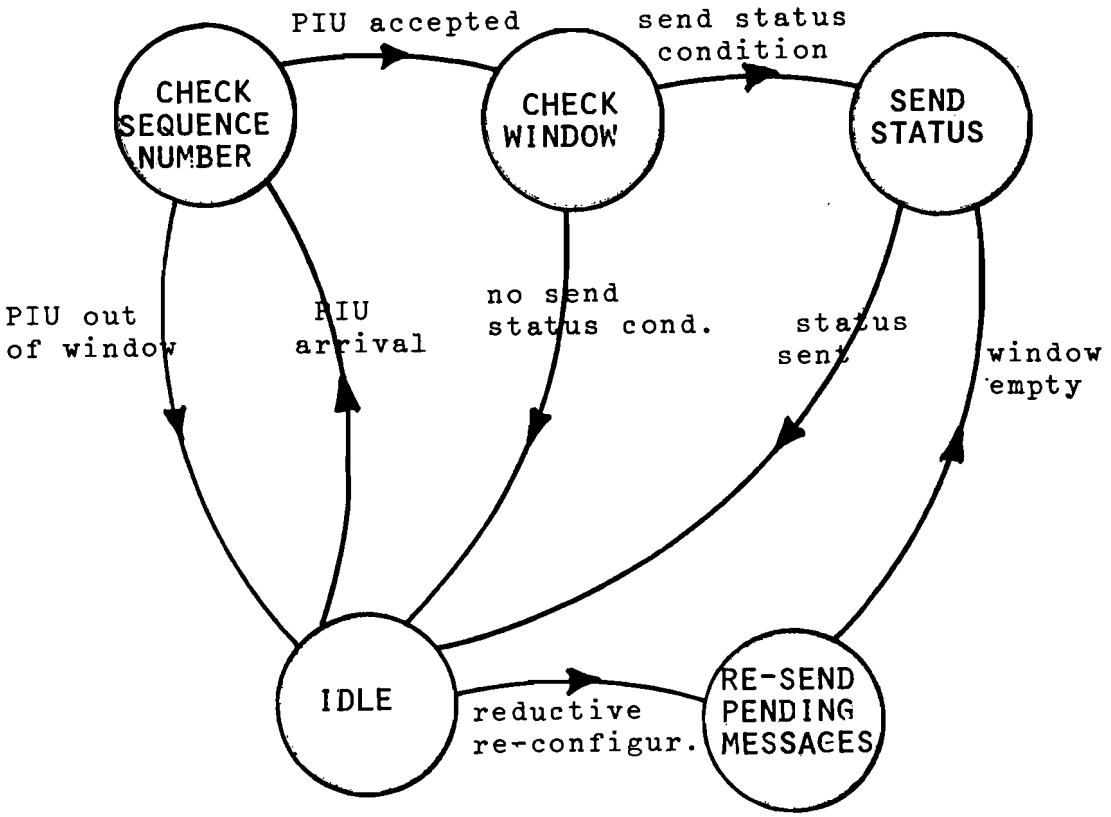


Figure 19 NSM Receiver

b) The BIND request puts the LU in "BIND REQUEST STATE". The requested LU is identified by the location (Node and Host) where it resides, and by the name of the Application to which it pertains. Moreover, the BIND request has to provide the password to access the requested LU

and, in case, a message for the Application owing the LU (for a further check of the connection possibility).

c) When a LU enters the "BIND REQUEST" state, a network address is assigned A LCT is arranged, and a message is built for the NSM (Session Service) of the location where the requested LU is. This "BIND message" has to contain the identifier of the requesting LU (network address) and owing Application (name), the symbolic identifier of the requested LU (Application name), the authorization password, and, in case, a message for the requested Application.

d) When receiving a "BIND message", the NSM (Session Services) has to check if there is a LU in "INVITE" state, with the requested identifier. If no one is found, a negative response is prepared and sent back to the originating NSM (Session Services). Otherwise, the authorization password is checked. If the provided password is incorrect, a negative response is prepared, and sent back to the originating NSM (Session Services). If the provided password is correct, the session can be established. A further control has to be made on the eventual request of an authorization message for the Application owing the LU. If this message is required, and if it is provided by the "BIND message", the requested LU enters into the "BINDING" state, the INVITE request is closed and the authorization message is forwarded to the Application If no authorization message is required, a positive response is prepared, and sent back to the originating NSM, and the requested LU enters into the "LCT ACTIVE" state.

e) When in "BINDING state, the LU is waiting for the Application response to the authorization message. If the connection is rejected, a negative response is prepared and sent back to the originating NSM; the LU returns to the "INVITE" state, or the "IDLE" state, depending on the Application reject code. If the connection is accepted, a positive response is prepared (including the, in case Application response message), and sent back to the originating NSM (Session Services) and the LU enters into the "LCT ACTIVE" state.

f) When in "BIND REQUEST" state, the receipt of a negative response causes the releasing of the reserved network address, and LCT, and the closing in exception of the BIND request. The LU returns to the "IDLE" state.

g) When in "BIND REQUEST" state, the receipt of a positive response closes successfully the BIND request (passing to the Application the, in case, response message), completes the LCT and puts the LU in the "LCT ACTIVE" state.

h) When in "LCT ACTIVE" state, the LU to LU session is established, and the activity will be made under the direct SH control.

i) When in "LCT ACTIVE" state, the UNBIND request causes the releasing of the LCT, and of the corresponding network address. An "UNBIND message" is prepared to be sent to the counterpart NSM (Session Service) in order to clear the Logical Channel. The LU returns to the "IDLE state under the direct NSM control.

l) When in "LCT ACTIVE" state, the receipt of an "UNBIND message" causes the releasing of the LCT and of the corresponding network address, the clearing of the pending activity on the logical channel, and the signalling to the Application owing the LU of the Session Close-down. The LU returns to the "IDLE" state under the direct NSM control.

When two Applications want to establish a session, first of all they have to agree on the role the single Application has to play primary (BIND request) or secondary (INVITE request), in order to avoid contention (both primary) or indefinite wait (both secondary). Furthermore, there is the problem of the synchronization, that is, the secondary LU has to be set up before the arrival of the primary LU request message, otherwise the BIND request will be refused for counterpart unavailability. The synchronization recovery could be in charge of the Application that wants to play the primary role, but there is not a criterium which may help in recovering such a synchronization lock. The only possibility is to try and to try again until the BIND operation is successfully closed.

In order to make the synchronization in opening a session easier for the Application, two options have been associated to the BIND and INVITE operations.

For the INVITE operation, there is the possibility to specify the BROADCAST option. When the LU enters into the "INVITE" state, if the BROADCAST option is in effect, a broadcast message is sent to all the NSMs (or to a specific NSM) in the network, carrying the information of the present availability of the LU. For the BIND operation, there is the possibility to specify the WAIT option, that is, in case of BIND failure due to counterpart unavailability, the LU is put into the "BIND WAITING" state, and where a "broadcast message" arrives with the indication that the previous requested LU is now available, the BIND request message is automatically resent, and the LU returns to the "BIND REQUEST" state.

The complete Session Service Protocol can be described, as usual, using the state diagram technique.

a) When in IDLE state, and a BIND request is received, the LU is put into the "BIND REQUEST" state, a LCT is reserved, a network address is assigned, and a "BIND request" message is sent to the NSM where the requested LU resides.

b) When in IDLE state, and an INVITE request is received, the LU is put into the "INVITE" state, a LCT is reserved, a network address is assigned, and the LU is identified by the owner Application name.

c) When in BIND REQUEST state, a negative response (including the counterpart unavailability with the NOWAIT option) puts the LU into the IDLE state. The BIND request is closed in exception.

d) When in BIND REQUEST state, a positive response puts the LU into the "LCT ACTIVE" state. The LCT control passes to the SH. The BIND request is closed successfully.

e) When in BIND REQUEST state, an unavailability response (if the BIND option WAIT has been specified) puts the LU into the "BIND WAITING" state. The BIND request is not closed.

f) When in BIND WAITING state, a broadcast message referred to the previously requested LU puts the LU back into the "BIND REQUEST" state.

g) When in "INVITE" state, and a BIND message is received, the LU is put into the "BINDING" state.

h) When in BINDING state, and the request is invalid (bad password, no "BIND message" for the owing Application) a negative response is sent back and the LU returns into the INVITE state. The INVITE operation is not closed.

i) When in BINDING state, and the request is valid, the "BIND message" carried by the BIND request is forwarded to the Application (if required). The LU remains in the BINDING state, waiting for an Application response, and the INVITE operation is closed.

l) When in BINDING state, and a reject response has arrived from the owning Application a negative response is sent back to the requesting NSM, and the LU is put into the IDLE state, or into the INVITE state depending on the reject option chosen by the Application (RESET or CONFIRM, respectively). If the LU returns into the INVITE state, the INVITE operation is reopened.

m) When in BINDING state, and the request is valid, and no more "BIND message" is required, the LU is put into the "LCT ACTIVE" state. The LCT is completed and activated, and the LCT control passes to the SH. The INVITE request is closed successfully. A positive response is sent back to the requesting NSM.

n) When in BINDING state and an Accept response has arrived from the owing Application, a positive response is sent back to the requesting NSM, the LU is put into the "LCT ACTIVE" state. The LCT is completed and activated, and the control passes to the SH.

This state diagram is depicted in the flowchart of Figure 20.

4.3.4.2 Mailing Service. The Mailing Service provides a LU with the possibility to send and to receive messages (letters) to and from another LU in the Network.

The Mailing Service is accessed via the MAIL request. A LU may use the MAIL request to send a message (letter), or to set up a mail box, in order to be able to receive a (some) message(s). Every mail box is identified by the name

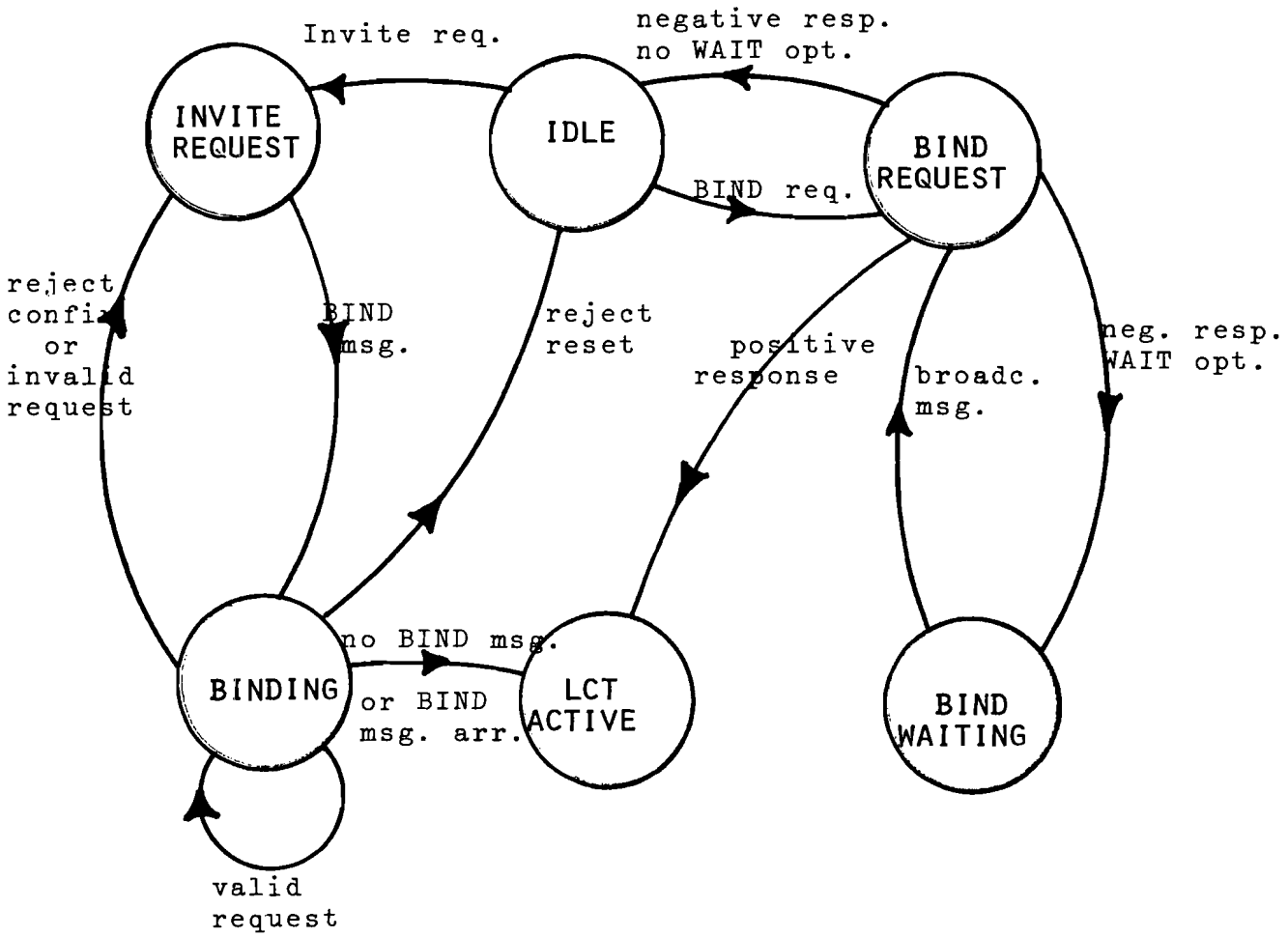


Figure 20 Session Service Protocol

of the Application owing the related LU, and by a further identifier (optional). No more than one mail box may be associated to a particular LU. A mail box remains active until a mail message arrives, closing the mail receive operation.

When used to send a message, the MAIL request has to besides the message text, provide the full address identifier (location, Node and Host, and the mail box identifier, short or extended). The delivery of the message may be guaranteed or not, depending on the MAIL request options The message delivery is guaranteed when the MAIL request is considered closed only when the message acknowledge (positive or negative) is received back. The message delivery is not guaranteed when the Mail request is considered closed as soon as the message leaves the originator Node and no acknowledge is expected. For what we said, it is clear that a MAIL Service protocol (a very simple protocol) had to be designed. The basic rules of this protocol are in what follows.

Through a LU, two kinds of requests may be presented to the Mailing Service: the Mail-Send request and the Mail-Receive request. Between the Mailing Services, two kinds of information may be exchanged: a mail message or a mail acknowledgement.

a) When a mail message and no mail box with the required address is present, a negative acknowledgement is sent back or the message is simply ignored, depending on the mail message option (guaranteed or not). In both cases, the mail message is discarded.

b) When a mail message arrives and the required mail box is found, the mail receive operation is closed, and a positive acknowledgement is sent back, if required (mail message guaranteed).

c) When the Mailing Service is unable to forward the message, because the destination is unreachable or invalid, a negative response closes the mail-send request.

d) When a mail message leaves the originator node, and no guarantee has been required, the mail send request is closed positively

e) When a mail acknowledgement arrives, the mail send request (if guaranteed) is closed negatively or positively, depending on the acknowledgement code.

4 3 4.3 Inquiry Service The Inquiry Service provides a LU with the ability to inquire about the structure and the status of the network environment. The Inquiry Service is accessed via the INQUIRE request. A LU may use the INQUIRE

request to inquire about:

a) the status (reachable or unreachable) of the specified Host and, if reachable, the number of intermediate Nodes on the presently shortest path;

b) the status (reachable, unreachable, available or not available) of a specified Application. If reachable, the number of the intermediate Nodes on the presently shortest path. If available, the total number of opened ports (LUs) of the Application, its total number of LUs in INVITE state and in session (LCTs).

c) the status (reachable or unreachable) of the specified Host. If reachable, the number of intermediate Nodes on the presently shortest path, the total number of opened ports (LUs), the number of LUs under NSM control, the total number of sessions, the number of sessions between the requiring Host and the addressed Host, the presently exchange rate (number of bytes sent and received per second).

The Inquiry Service is not implemented at the present state of RPCNET, but the software is full-compatible with such a service, in all the RPCNET implementations.

4.3 4.4 Network Operator Service. The Network Operator Service provides the local operator with the ability to monitor the Network activity. The way in which such a service can be implemented depends mainly on the Operating System under which the RPCNET software has to be implemented, so that we can only suggest a set of commands which should be made available for the Network Operator:

- Start of Node Network activity
- Shut down of Node Network activity
- Activation or de-activation of Network lines
- Inquiry about Network link(s) status
- Inquiry about LUs status
- Inquiry about Session status

- Inquiry about Network topology status
- Forcing a Session close down
- Forcing a LU release
- Sending of messages to other Network operators

4.4 The Network Access Controller

The Network Access Controller is the most external sub-layer of the Communication software, so that its design depends mainly on the Operating System in which it has to run.

The Network Access Controller (which is also called Call and Return Interface) performs a first set of controls on the user issued operation, before passing it down to the NSM, or to the SH depending on the operation code.

The NAC can be called by Netusers via a branch-and-link instruction, or a Supervisor Call instruction depending on the Operating System For example, in the VM implementation, the NAC call is performed by a set of routines which are called by the Netuser via a branch-and- link instruction, while in the OS implementation, a new SVC code has been added to the system, to perform the NAC call

The NAC call is, usually, the last istruction of any RNAM macro, as it will be mentioned later.

5 THE APPLICATION LAYER

5.1 Introduction

The Application Layer is the most external layer of the RPCNET; it communicates with the other layers by using an access method, called RNAM (Reel Network Access Method) which consists on a set of macros, and/or a set of high-level language subroutines. We will speak about RNAM later, now, let us say something about Applications, in general.

An Application is a program, or a set of programs, splitted in two parts, which reside on different computers, and communicate each other using RNAM. An Application can be a user-written program or an already existing system program, which has to be modified in order to introduce the Network concepts. For example, a system "spool" handler can be modified in order to take into account the possibility to send "spool" files not only to the local output devices, but also to different remote computer devices. A typical user-written Application could be simply a program which needs data residing on another computer: if there is another program, and a protocol of communication was established, the two programs can communicate each other.

As it is impossible to see that a computer network exists, without providing any facility, such as a spool file transfer, or something like this, it is obvious that some Applications should be developed by the systemists who develop the Network software, so that two aims can be achieved:

a) testing of the Network software (the best way to test a software is to make it working)

b) providing a first kind of "service", which allows users to enter the Network philosophy, and stimulates the study and the developing of other Applications.

5.2 RNAM

RNAM, the Reel Network Access Method, is a functional unit that enables an Application program to establish and use ports (LUs) on the Communication System (Communication Layer, plus Interface Functions Layer).

The services provided by RNAM are:

a) macro function services;

b) data flow control services;

c) Logical Unit handling services.

The macro function services are all the services necessary to analize and execute the Application program requests made via a one of the supported macro intructions.

The data flow control services are provided by the data flow control protocol support (SH). This protocol is used to handle data structures as a chain, or to manipulate the state conditions such as "send" or "receive" state of the associate Logical Unit.

The Logical Unit handling services consist in:

a) Logical Unit set-up and release

b) Logical Unit BIU traffic handling

c) Logical Unit exception condition handling.

5.2.1 Characteristics of RNAM

The RNAM characteristics are:

a) application program basic interface

b) synchronous handling of the requests

c) asynchronous handling of the requests

d) application program macro instructions

e) operation and asynchronous activity handling

f) data flow control

5.2.1.1 Application program basic interface. The RNAM interface versus the Application programs is a BAL (Basic Assemble Language) type interface. The program access RNAM either via a Branch and Link instruction or a Supervisor Call, depending on the implementation and/or the Operating System.

All the requests are made using a request code and a parameter list, that specifies the characteristics of the request itself. An Application program using RNAM can be described as a "single-thread" program, that is a program capable of handling only one port (LU) at a time, or as a "multi-thread" one, that is, capable of processing the requests of many LUs concurrently. So that, in general, a "single-thread" program requests synchronous operations and waits until each synchronous operation is completed, while a "multi-thread" program requests asynchronous operations, and continues processing on behalf of other LUs while waiting for an operation in a particular LU to be completed.

5.2.1.2 Synchronous handling of the requests. In a synchronous program, the operations are performed in a serial pattern. A request for a single operation (for example a SEND or a RECEIVE) means that RNAM returns control to the next sequential instruction in the program only after the requested operation is completed. This means that the execution of the Application program is halted until RNAM determines that the operation has been completed. A synchronous operation is depicted in Figure 21.

5.2.1.3 Asynchronous handling of requests. In an asynchronous operation, RNAM returns control to the next sequential instruction as soon as RNAM has accepted the request, not when the requested operation has been completed. Accepting a request consists on screening the request for logical errors, and scheduling the requested operation. While the operation is being completed, the Application program is free to initiate other LU processings.

When an asynchronous operation is specified, there are two ways in which RNAM can notify the Application program that the operation has been completed. If the Application program associates an Event Control Block (ECB) with the request, RNAM posts the ECB when the operation has been completed.

Alternatively the Application program can associate an exit-routine with the request. When the operation is completed RNAM schedules the routine. In Figure 22 and

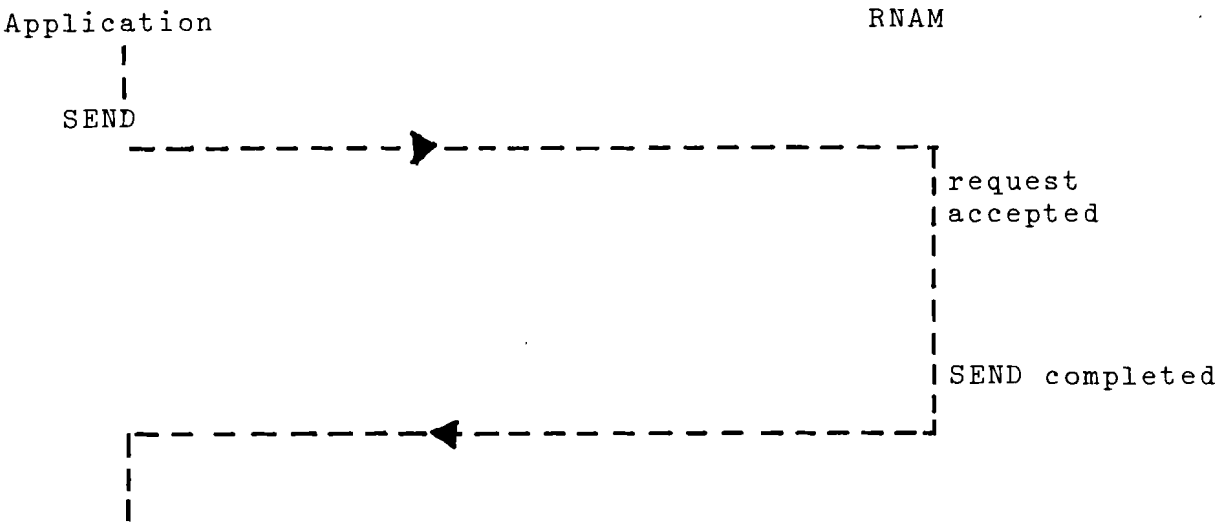


Figure 21 Synchronous requests handling

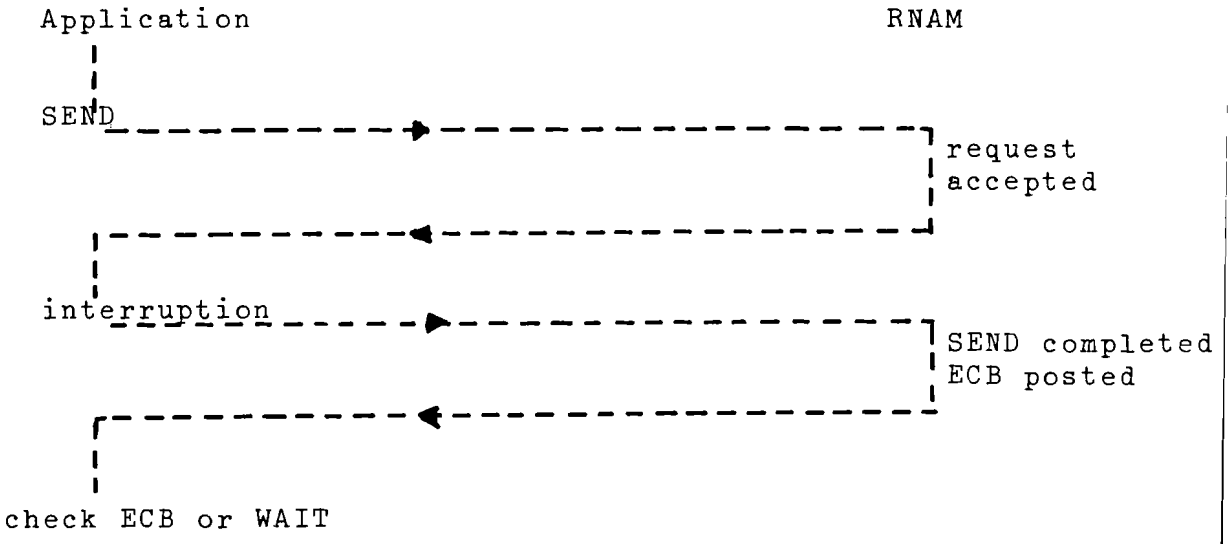


Figure 22 Asynchronous request with ECB

Figure 23 it is possible to see examples of asynchronous processing in an Application program using ECB or an exit routine.

5.2.1.4 Application program macro instructions. RNAM provides the Application program with a set of macro instructions to perform all the allowed Network activities Basically, the macro instructions can be divided in five classes, as follows:

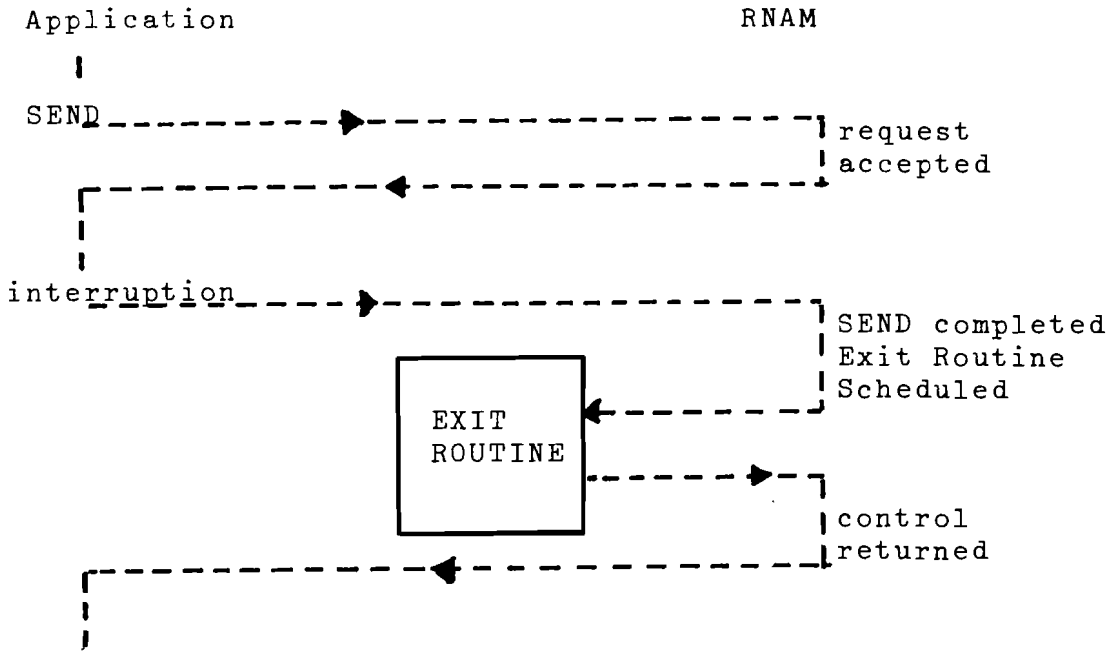


Figure 23 Asynchronous request with Exit-routine

a) macros to set-up and release LUs
OPENLU The OPENLU macro instruction acquires port(s) on the Communication System.
CLOSELU The CLOSELU macro instruction releases the LUs acquired

b) connection macros
BIND The BIND macro instruction is a request for connecting the referred LU to a LU of the specified Application (in the Network environment), in order to establish a session.
INVITE The INVITE macro instruction is used to place the corresponding LU in INVITE state, waiting for the connection with another LU that wants to establish a session issuing the BIND macro instruction.
UNBIND The UNBIND macro instruction is a request to break-up the connection between the referred LU and its counterpart in the established session.

c) I/O macros
SEND The SEND macro instruction is a request to transfer data from the requesting LU to its counterpart in the established session.
RECEIVE The RECEIVE macro instruction is a request to transfer into the Application program area data coming, on the referred LU, from the counterpart LU during the session
BREAK The BREAK macro instruction enables the Application program to break-up the send-

receive data flow, allowing the Application to send asynchronous data to its counterpart in the session, whatever its role (sender or receiver) may be.

d) support macros
TESTLC The TESTLC macro instruction enables the Application program to test and to get information on the logical path from the referred LU to its counterpart LU in the established session.
CANCEL The CANCEL macro instruction enables the Application program to cancel (under certain conditions) the operation pending on the referred LU.

e) Network Services macros
INQUIRE The INQUIRE macro instruction enables the Application program to inquiry about the Network environment, using the services offered by the Communication System.
MAIL The MAIL macro instruction enables the Application program to send messages to another Netuser, using the mailing services offered by the Communication System.

5.2.1.5 Operation and asynchronous activities handling. As far as RNAM is concerned, every operation might be considered completed as soon as it has been scheduled for the execution on the Communication System. Although due to the particular nature of the utilized system (a Computer Network), some of the operation must be considered completed only when a feed-back information is provided to RNAM by one or more of the Communication System components

A distinction can be made between immediate and non-immediate operations like in a standard I/O system. In parallel with a standard I/O system, RNAM may consider completed either the operation (for immediate operation request), or its role (for non-immediate operation request), as soon as it receives a positive condition code on its operation execution request (the SIO instruction for the I/O system). At this point, the operation control passes to the Communication System component (Channel, Control Unit and Device for the I/O system).

One or more of these components will subsequently provide the ending status of the operation to the requesting module (RNAM). Depending upon the type of the request, the ending condition might be an unique collection of intermediate ending status (Channel end, Control Unit end and Device end for the I/O system), or can appear as

subsequent and separate ending states (for instance, first Channel end then Control Unit end, and finally Device end for the I/O system).

In the following Figure 24 a scheme of the handling of the single operations, as far as the ending conditions is concerned, is presented. The only immediate operations are related to the MAIL and UNBIND macros. In the figure only normal completion is considered.

Besides the standard operation activity, an asynchronous activity versus the Application program must be considered in RNAM. The asynchronous activity is introduced in RNAM, through the LUs, by the Communication System, and can be divided in four classes:

a) Exception messages: not directly required by the Application program, the exception message represents the new logical status of a LU, or a LU to LU logical path, when modified by an exception condition.

b) Exception responses: directly required by the Application program when it uses the SEND macro instruction and wants to have only the, in case, negative feed-back of the operation.

c) Break data: not directly required by the Application program, break data represent an information, coming from the connected Application during a session, that breaks-up the present sender-receiver data flow.

d) Mail: because of the mailing service offered by the Communication System, it is possible at any time to receive mail messages from other Network users. This activity is not directly required by the Application program, and can be suspended on direct Application program request.

5.2.1.6 Data flow control. The data flow control service offered by RNAM is limited to the implementation of a data flow control protocol based on the service offered by the Communication System to each LU. This protocol is used to handle such data structures as "chains", or to manipulate state conditions such as "send" or "receive" state of the associated LUs during a session.

The data flow control protocol does not perform any transformation function on the "end user" request, but assists "end users" in controlling the flow of the requests.

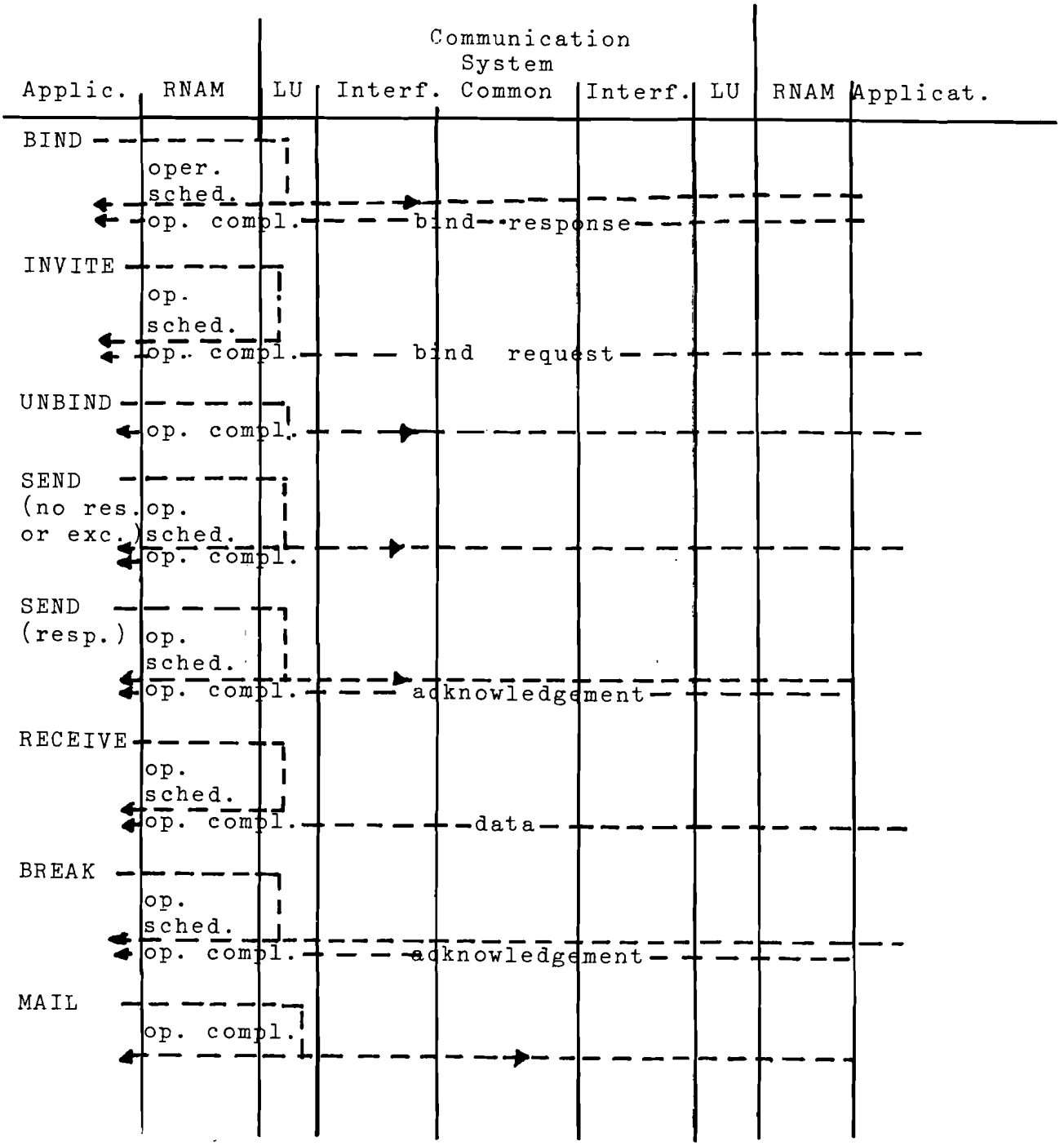


Figure 24 RNAM handling of the operations

5.3 Improved Access Method based on RNAM

Normally, every access method for data transfer handling provides not only a set of basic macros as RNAM does, but also a set of support macros for building and handling of control blocks and parameter list.

This set of macros could be operating system dependent at least at the implementation if not at the formal level. An improved access method based on RNAM is presented in this section, in which only a few macros have been added in order to achieve, at least at the formal level, an operating system independence. The fact that an Access Method had to be implemented on different operating systems was taken in consideration.

Beside this, a different implementation migth require some new macros beside the basic set presented in this section, or a different use of the described macros. In any case, the following description must be considered only a suggestion on how to implement an RNAM based Access Method.

5 3.1 General characteristics of the Access Method

All the RNAM characteristics have to be maintained in the Access Method moreover the macro function services of the Access Method have to be improved.

The new services comprise an authomatic building and handling of all the necessary control blocks and parameter lists, as well as on user oriented operation checking and error handling. All these functions have to be provided by the Access Method via macro instructions and might be tailored to the particular operating system.

5.3.2 Macro instruction structure

In the following paragraph a formal structure of the Access Method supported macro instruction is given.

All the macro instructions are divided in six classes:

a) Declarative macros
   RPL Request Parameter List
   LUCB Logical Unit Control Block
   EXLST Exit-routines List

b) Manipulative macros
   GENCB Generate a Control Block
   MODCB Modify a Control Block

c) LUCB based macros
   OPENLU Open Logical Unit
   CLOSELU Close Logical Unit

d) RPL based macros

Connection macros
   BIND
   INVITE
   UNBIND

Data transfer macros
   SEND
   RECEIVE
   BREAK

e) Macros that support connection or I/O
   CHECK
   CANCEL
   EXECRPL
   TESTLC

f) Macros for the Network environment
   INQUIRE
   MAIL

5.3.3 Macro Instructions

Each macro instruction description contains a three column table that shows how the macro intruction has to be coded. Since macro instructions are coded in the same format as assemble intructions, the three columns correspond to an assembler intruction label, operating code and operand fields. All the operands are keyworded or positional operands, depending on the particular implementation.

Keyword operands consist in a first character string (the operand keyword), an equal sign, and a single or multiple operand value. Keyword operands do not have to be coded in the order shown in the operand column, and must be separated by commas. If more than one value can be coded after a keyword, the parentheses are required.

A notation scheme is illustrated in the following, to show how, when and where operands can be coded. The notational symbols are never coded.

- an exclamation point (!) means "exclusive or"
- vertical arrows (^) are used to group alternative operand values One of the alternative values enclosed within the arrows must be chosen.
- a value enclosed between two plus signs (+) means that if no value is selected for that operand, that value is assumed as default value.
- percentages (%) denote optionals operands.
- an ellipsis (...) indicates that whatever precedes it (either an operand value, or an entire operand) can be repeated any number of times.
- parentheses, equal signs and uppercase characters must be coded exactly as shown in the operands column. Lowercase words represent values that the user must supply.

LUCB Logical Unit Control Block

The LUCB identifies the Logical Unit to be obtained from the communication system. Every Application port must have a LUCB; an Application may have more than one LUCB. A LUCB macro instruction causes a LUCB to be built during the program assembly. The LUCB can also be built during program execution using the GENCB macro instruction, and can be modified during program execution by means of the MODCB macro instruction, see Figure 25.

Name	Operation	Operands
label	LUCB	%,BREAKLN= break area length %,DATALEN= max data transfer %^,ECB= event contr. block addr. ^,EXIT= Exit routine address %,EXTLST= Exit routine list addr.

Figure 25 LUCB macro instruction

BREAKLEN = break area length
Indicates the length (in bytes) of the area (in LUCB) destinated to accomodate the incoming asynchronous data generated by a BREAK macro issued by the counterpart during a session. The maximum number which can be specified is 128. If this operand is omitted, the BREAKLEN field is set to Ø, and no incoming break data will be accepted.

DATALEN = max record length for data transfer
Indicates the maximum length (in bytes) that will be specified for data transfer requests related to this LU (during a session).

ECB = event control block address
Indicates the location of an event control block to be posted by RNAM when a not requested asynchronous activity (break data coming or exception response arrived) has to be signalled on the Logical Unit related to this LUCB. The ECB can be any word of addressable storage. The ECB field and the EXIT field share the same LUCB field. The ECB-EXIT field is used in this manner:

- If ECB = address has been specified, RNAM uses the field as the address of an external ECB.

- If EXIT = address has been specified, RNAM uses this field as the address of the LUCB Exit-routine and schedules the routine as indicated below (under EXIT operand).

- If neither ECB = address, nor EXIT = address has been specified, RNAM uses this field as an internal ECB. RNAM does not clear the ECB. Users of ECB must be sure to clear the ECB once the ECB itself has been posted.

EXIT = LUCB Exit-routine address
Indicates the address of a routine to be scheduled when an asynchronous activity has been required on this LU When the routine receives control, some general purpose registers will contain the following:

- Reg X : the address of the LUCB associated with the LU whose asynchronous activity has caused the LUCB Exit-routine to be entered;

- Reg Y : the address in RNAM program to which the LUCB Exit-routine must branch when it has finished (return address);

- Reg Z : the address of the LUCB Exit routine itself. No register save area is provided upon invocation of the LUCB Exit-routine.

EXTLST = exit routine list address
Indicates the address of a list of Exit-routines to be entered when particular events are recognized on the LU by RNAM.

All the LUCB fields above are fields set by the Application program. The fields described below are set by RNAM, and can be tested by the Application program.

CONTROL After an asynchronous information has been received, CONTROL is set to one of the following values:
BRKREC Break data received. See BRKAREA and ASYLEN fields.
LUSTAT Logical Unit status indicator received, check LUSENSEI.
EXCRESP Exception response received. See RESPNUM field for the number of request which

this response is referred to; and the LUSENSEI field for further information.

ASYLEN The ASYLEN field is set by RNAM when an asynchronous input operation is finished, to indicate the length of the data just received.

RESPNUM The RESPNUM field is set by RNAM when an asynchronous response (exception response or network inquiry response) is received. This field contains the sequence number of the request the response is referred to.

LUSENSEI When an exception message (Logical Unit status) or an exception response is received, the LUSENSEI field indicates the presence of a system error code.

LUSENSEM The LUSENSEM field contains the system indicator modifier bits.

BRKAREA This field, whose length (in bytes) is specified in the BREAKLN field, will contain the incoming break data address.

RPL Request Parameter List for a Logical Unit

Every request that an Application program makes for connection, data transfer or Network services operations, must refer to an RPL. A Request Parameter List (RPL) is a control block used by the Application program to describe the request it makes to RNAM, and it is related to a particular Logical Unit by the LUCB field. The RPL macro instruction builds an RPL during assembly. An RPL can be generated also during program execution with the GENCB macro instruction Request for RPL modification can be made as part of a request macro, or by the MODCB macro instruction, see Figure 26.

Name	Operation	Operands
label	RPL	LUCB= LUCB address %,AREA= data area address %,AREALEN= data area length %,RECLEN= data length %,OPTION= +SYN+!ASY %^,ECB= event contr. block addr. ^,EXIT= Exit routine address %,CHNGDIR= YES!+NO+ %,REQRESP= !EXC!DEF!+NO+ %,DESTID= destination identifier %,APPLID= application identifier %,PASSW= authorization keyword

Figure 26 RPL macro instruction

LUCB = LUCB address. Associates the request that will use this RPL to a LUCB

AREA = data area address. When used by a SEND or RECEIVE macro instruction, AREA indicates the address of an area in program storage from which data is to be written or into which data is to be read. When used by an INVITE macro instruction, with option BINDMSG = YES, AREA indicates the address of an area where the data obtained by the LU which wants to connect the Application, is to be placed. When this operand is omitted, AREA field is set to Ø.

AREALEN = Data Area Length (only for RECEIVE). Indicates the length (in bytes) of the data area identified by the AREA operand. The AREALEN operand is meaningful only for input operations. For the RECEIVE macro instruction, AREALEN = Ø means that no input data area is available. If

omitted, AREALEN is set to Ø. RECLEN = Data Length (only for SEND). When used by a SEND or BREAK operation, RECLEN indicates the length (in bytes) of the data to be transferred. For RECEIVE operation, the RECLEN operand has no meaning, but this field is set by RNAM when the input operation is completed, to indicate the length of data received. When a RECEIVE operation is completed and excess data is detected, RECLEN contains the total length of the originally available data. If omitted, this field is set to Ø.

OPTION = Option Code. Indicates options that are to affect the request made using this RPL.

SYN!ASY Indicates whether RNAM should synchronously or asynchronously handle any request made using this RPL.

SYN Synchronous handling means that, when a request is made, control is not returned to the Application program until the requested operation has been completed (successfully or otherwise). The Application program should not use the CHECK macro instruction for synchronous requests; RNAM automatically performs this checking (which includes clearing the ECB). When control is returned to the Application program, a general purpose register will contain the completion code.

ASY Asynchronous handling means that after RNAM schedules the requested operation, control is immediately passed back to the Application program. When the event has been completed, RNAM makes one of the following actions:

- If the ECB address is specified for the RPL, RNAM posts a completion indicator in the event control block indicated by this operand. If neither an ECB nor an EXIT address is specified in the RPL, the ECB field itself is used as an event control block. The Application program should issue a CHECK (or a system WAIT) macro to determine whether the ECB has been posted.

- If the EXIT operand is in effect, RNAM schedules the Exit-routine indicated by this operand.

Note: After an asynchronous request has been accepted, and before that request has been completed, the RPL being used by the request must not to be modified. A modification during this interval could cause RNAM to be unable to complete the request in a normal manner, which in turn would cause RNAM to terminate the Application program.

ECB = event control block address Indicates the location of an event control block to be posted by RNAM when the request associated with this RPL is completed The ECB field and the EXIT field share the same RPL field. If asynchronous handling of the request has been specified (ASY option in the RPL) the same happens as it did for the ECB parameter in LUCB. If synchronous handling (SYN option in the RPL) has been specified, the ECB-EXIT field is used as follows:

- If ECB = address has been specified, RNAM uses that field as an the address of an external ECB.

- If EXIT = address has been specified, RNAM uses the field as an internal ECB, thus destroying the Exit routine address.

- If neither ECB = addres, nor EXIT = address has been specified, RNAM uses this field as an internal ECB.

RNAM clears internal ECBs when it begins processing any RPL-based macro, and when the RPL is checked. RNAM clears external ECBs only when the RPL is checked. RPL checking is made by RNAM at request completion for synchronous handling, and it is made by the user issueing CHECK for asynchronous request handling). Users of external ECBs must therefore be sure that the external ECB is cleared before the next RPL-based macro is issued.

EXIT = RPL Exit Routine Address. Indicates the address of a routine to be scheduled when the request represented by this RPL is completed. If the SYN option has been specified, the Exit-routine is not used; if specified, the address is overwritten before the synchronous request completes (RNAM uses this field as an internal ECB in this situation). The RPL Exit-routine is scheduled only if asynchronous handling of the request has been specified. When the routine receives control, three general purpose registers contain the following:

Rx the address of the RPL associated with the request whose completion has caused the RPL Exit routine to be entered.

Ry the address (in RNAM program) to which this routine must branch when it is through processing (return address).

Rz the address of the RPL Exit-routine itself.

REQRESP = (EXC!DEF!+NO+) When a SEND macro is issued, the REQRESP field designates the type of response desired for this request.

EXC Only exception response expected. Only in case of a Network error a response will be sent to this request.

DEF Definite response expected. A response is expected in any case to complete the request.

NO Neither positive nor negative response is expected for this request.

CHNGDIR = (YES!+NO+). When a SEND macro is issued, the CHNGDIR field designates if the Change Direction Indicator is to be set on (YES) or off (NO).

YES the Change Direction Indicator is set to signal the port that the transmitting LU ceases transmitting on its own initiative, and prepares to receive.

NO no change in the actual role of the transmitting LU.

DESTID = Destination Identifier. This field is used by the BIND macro instruction to indicate the Network location where the Application to be bound is. The DESTID operand contains either the address of the identifier, or the identifier itself. The DESTID field contains directly the identifier.

APPLID = Application Identifier. This field is used by the BIND macro instruction to indicate the identifier of the Application to be bound. The APPLID operand contains either the address of the identifier, or the identifier itself The APPLID field contains directly the identifier.

PASSW = Authorization Keyword. This field is used by the BIND macro instruction to indicate the keyword necessary to access the requested Application. The PASSW operand

contains either the address of the keyword or directly the keyword itself. The PASSW field contains directly the keyword.

All the RPL fields described above are fields set by the Application program. The fields described below are set by RNAM at the completion of the request.

RTNCD : A general return code returned by all of the RPL based macro instructions

FDBK (Feedback): Specific error return code returned by all the RPL based nacros that are accepted by RNAM but are not completed successfully.

REQ : A value returned by all of RPL based macros, except EXECRPL and CHECK, that identifies the type of macro instruction.

CHAIN : When a RECEIVE macro has been completed, the CHAIN field indicates the message's relative position in the Chain. The possible settings are: FIRST, MIDDLE, LAST or ONLY.

RESPONSE: The RESPONSE field indicates further details when a SEND or BREAK operation has been ended with an exception response (SEND operation with Definite Response required).

GENCB Generate a control block

The GENCB macro instruction builds a LUCB or an RPL. The advantage of using the GENCB macro is that the control blocks are generated during the program execution. GENCB not only builds the control block during program execution, but can also build the control block in dynamically allocated storage.

The GENCB user specifies the type of control block to be built, and the contents of its fields. The operands used to specify the field contents are exactly the same as those used in the macro instruction that builds the control block, see Figure 27.

Name	Operation	Operands
label	GENCB	BLK= ^RPL!LUCB^ %,KEYWORD= value%... %,COPIES= quantity %,WAREA= work area address %,LENGTH= work area length

Figure 27 GENCB macro instruction

BLK= RPL!LUCB: indicates the type of control block to be generated.

KEYWORD= Value: indicates a control block field and the value that is to be contained or represented within it. For "keyword" can be coded any keyword that can be used in the macro instruction corresponding to the BLK operand.

COPIES= Quantity: indicates the number of control blocks to be generated. The copies are identical in form and contents; they are placed contigously in storage. If this operand is omitted, one control block is built.

WAREA= Work Area Addres: indicates the location of the storage area in the Application program where the control block is to be built. If this operand is specified, the LENGTH operand must be specified too. If the WAREA and LENGTH operands are omitted, storage area can be dynamically obtained (it depends on the operating system) via some system facility, and the control block is built.

LENGTH= Work Area Length: indicates the length (in bytes) of the storage area designated by the WAREA operand.

MODCB Modify a control block

MODCB modifies the contents of one or more fields in a control block. MODCB works with control blocks created either by declarative macros, or by the GENCB macro. The user of the MODCB macro instruction indicates the location of a LUCB or RPL, the fields within that control block to be modified and the new values that are to be placed or represented in these fields. Any field whose content can be set by the LUCB or RPL macro instructions, can be modified by the MODCB macro. The operands used to do this are the same as those used when the control block is created, see Figure 28.

Name	Operation	Operands
label	MODCB	^LUCB= LUCB address ^RPL= RPL address %,Field name= new value%,...

Figure 28 MODCB macro instruction

LUCB= LUCB address

RPL= RPL Address: indicates the type and location of the control block whose fields are to be modified. Field Name= New Value: indicates a field in the control block to be modified, and the new value that is to be considered or represented within it

OPENLU

The OPENLU macro instruction requires RNAM to acquire one or more Logical Units on the Communication System.

The characteristics of the required Logical Units are contained in the referred LUCB, see Figure 29.

Name	Operation	Operands
label	OPENLU	LUCB= LUCB address %,...

Figure 29 OPENLU macro instruction

LUCB= Address: indicates the address of the LUCB that contains the characteristics if the Logical Unit to be activated.

CLOSELU

The CLOSELU macro instruction requires RNAM to release one or more Logical Units previously acquired.

The LUs to be released are indicated by the referred LUCB, see Figure 30.

Name	Operation	Operands
label	CLOSELU	LUCB= LUCB address %,...

Figure 30 CLOSELU macro instruction

LUCB= Address: indicates the address of the LUCB of the Logical Unit to be released and de-activated.

BIND

The BIND macro instruction is a request for connecting the referred LU to a LU of the specified Application, in order to establish a session. The LU involved in the operation is identified by the LUCB field of the RPL.

The amount of data that can be transferred with the BIND message is limited. This limit is fixed by the Communication System and, in any case, it can not overcome the maximum length permitted for a packet by the Communication System, see Figure 31.

Name	Operation	Operands
label	BIND	RPL= RPL address %,RPL field name= new value %,...

Figure 31 BIND macro instruction

RPL= RPL Address: indicates the location of the RPL that describes the BIND operation.

RPL Field Name= New Value: indicates an RPL field to be modified, and the new value that is to be contained or represented within it.

Although any RPL operand can be specified, the following operands apply to the BIND macro instruction.

DESTID= Destination Identification: indicates the Network location where the Application to be bound is. The DESTID operand contains either the address of the identifier or directly the identifier itself The DESTID field contains direcly the identifier.

APPLID= Application Identification: indicates the identifier (1 to 8 alphanumeric characters) of the Application to be bound. The APPLID operand contains either the address of the identifier or directly the identifier itself. The APPLID field contains directly the identifier.

PASSW= Authorization Keyword: indicates the keyword (1 to 8 alphanumeric characters) necessary to access the desiderate Application. The PASSW operand contains either the address of the keyword, or directly the keyword itself. The PASSW field contains directly the keyword.

BINDMSG= YES!+NO+: indicates whether or not the Application program wants to send a binding message to the desidered Application.

AREA= Data Area Address: data contained at the location indicated by this field is to be used in the operation.

RECLEN= Data Length: indicates the length (in bytes) of the data to be transferred.

ACTION= +BIND+!REJECT!ACCEPT: indicates how the Application program wants to use the BIND macro instruction: to request a connection or to accept or refuse a connection.

BIND. The Application program wants to request a connection, the DESTID and APPLID fields are used to determine the counterpart in the desidered connection, the PASSW and BINDMSG fields are used to collect authorization information to be presented to the required Application

REJECT. The Application program wants to reject the connection request presented via a binding message. The BINDMSG indicates if the Application wants to send further information explaining the connection rejection.

ACCEPT. The Application program wants to accept the connection request presented via a binding message. The BINDMSG field is used to establish whether or not the Application wants to send further information about the accepted connection.

INVITE

The INVITE macro instruction enables the Application program to place the referred Logical Unit in "INVITE" state, waiting for the connection with another Logical Unit that wants to establish a session with the Application issueing the INVITE macro instruction. The involved Logical Unit is indicated by the referred RPL, see Figure 32.

Name	Operation	Operands
label	INVITE	RPL= RPL address %,RPL field name= new value %,...

Figure 32 INVITE macro instruction

RPL= RPL Address: indicates the location of the RPL that describes the INVITE operation.

RPL Field Name= New Value: indicates an RPL field to be modified, and the new value that is to be contained or represented within it

Although any RPL operand can be specified, the following operands apply to the INVITE macro instruction:

AREA= Data Area Address: the area at the location indicated by this field is to be used to place the coming binding message (if required).

AREALEN= Data Area Length: indicates the length (in bytes) of the data to be transferred.

UNBIND

The UNBIND macro instruction is a request to break-up the connection between the referred Application Logical Unit and its counterpart in the established session.

The involved LU is indicated by the referred RPL, see Figure 33.

Name	Operation	Operands
label	UNBIND	RPL= RPL address

Figure 33 UNBIND macro instruction

RPL= RPL Address: indicates the location of the RPL that describes the UNBIND operation.

SEND

The SEND macro instruction is a request to transfer data from the requesting LU to its counterpart in the established session.

The SEND macro instruction can be used only when the session has been established and only by the part to which the Communication System has assigned the "sender" role in the use of the Logical Channel.

The "sender" role is assigned, at the opening of the session, to that part which played the active role in the connection (using the BIND macro instruction) and can be exchanged only on initiative of the "sender" itself, that can pass this role using the Change Direction Indicator field. The involved LU is indicated by the referred RPL, see Figure 34.

Name	Operation	Operands
label	SEND	RPL= RPL address %,RPL field name= new value %,...

Figure 34 SEND macro instruction

RPL= RPL Address: indicates the location of the RPL that describes the SEND operation.

RPL Field Name= New Value: indicates an RPL field to be modified, and the new value that is to be contained or represented within it.

Although any RPL operand can be specified, the following operands apply to the SEND macro instruction:

AREA= Data Area Address: the data contained at the location indicated by this field are sent to the connected LU.

RECLEN= Output Data Length: the number of bytes indicated in this field is sent to the connected LU.

REQRESP= EXC!DEF!+NO+: this field designates the type of response desiderated for this request.

EXC Only exception response expected.

DEF Definite response expected.

NO No response expected.

CHNGDIR= YES!+NO+: this field designates if the Change Direction Indicator is to be set on (YES) or off (NO).

CHAIN= FIRST!MIDDLE!LAST!+ONLY+: indicates the relative position of the message within the chain currently being transmitted. ONLY means that the message is the sole element of the chain.

RECEIVE

The RECEIVE macro instruction is a request to transfer in the Application program area, data coming, on the referred LU, from the connected LU during a session.

The RECEIVE macro instruction can be used only when the session has been established, and only by the part to which the Communication System has assigned the "receiver" role in the Logical Channel utilization. The "receiver" role is assigned, at the opening of the session, to that part that played the passive role in the connection (using the INVITE macro instruction). The "receiver" role can be exchanged only on initiative of the "sender" (see the SEND macro instruction). The involved LU is indicated by the RPL field, see Figure 35.

Name	Operation	Operands
label	RECEIVE	RPL= RPL address %,RPL field name= new value %,...

Figure 35 RECEIVE macro instruction

RPL= RPL Address: indicates the location of the RPL that describes the RECEIVE operation.

RPL Field Name= New Value: indicates an RPL field to be modified, and the new value that is to be contained or represented within it.

Although any RPL operand can be specified, the following operands apply to the RECEIVE macro instruction:

AREA= Input Area Address: the data that will be received must be placed at the location indicated by this field.

AREALEN= Input Data Area Length: indicates the length (in bytes) of the data area identified by the AREA operand.

BREAK

The BREAK macro instruction enables the Application program to break-up the send-receive data flow, allowing the Application to send asynchronous data to its counterpart in the session, also if its role is "receiver"

The BREAK macro instruction can be used only when the session has been established. The amount of data that can be transferred is limited to the maximum length permitted by the Common Network. The involved LU is indicated by the LUCB RPL field, see Figure 36.

Name	Operation	Operands
label	BREAK	RPL= RPL address %,RPL field name= new value %,...

Figure 36 BREAK macro instruction

RPL= RPL Address: indicates the location of the RPL that describes the BREAK operation.

RPL Field Name= New Value: indicates an RPL field to be modified, and the new value that is to be contained or represented within it.

Only the following RPL operands can be specified:

AREA= Data Area Address: the data contained at the location indicated by this field are to be used in the operation.

RECLEN= Data Length: indicates the length (in bytes) of the data to be transferred

CHECK

The CHECK macro instruction enables the Application program to check the status of a requested operation, and, in case, waits for the completion of the operation itself.

When asynchronous handling has been specified for a request (ASY option in effect), the Application program receives control when the requested operation has been scheduled. A CHECK macro instruction must be issued for the RPL used for the request. CHECK should not be issued for synchronous requests.

When CHECK is used, the following action occurs:

- If the requested operation is not yet completed, CHECK suspends program execution until it is completed. If the RPL indicates an external ECB, or if the ECB-EXIT field is not set, CHECK returns the control to the Application program, when RNAM posts the ECB complete. CHECK clears the ECB before returning control. Users of external ECBs with asynchronous request handling must clear the external ECB (with CHECK or with assembler instructions) before the next RPL based macro is issued.

- If the operation completed with a logical or other error, CHECK causes the LERAD or ERAD exit-routine to be invoked, assuming that one is available.

This action also occurs when CHECK is issued in any RPL Exit-routine, see Figure 37.

RPL= RPL Address: indicates the location of the RPL to which the CHECK macro is to be referred.

Name	Operation	Operands
label	CHECK	RPL= RPL address

Figure 37 CHECK macro instruction

CANCEL

The CANCEL macro instruction enables the Application program to cancel the operation pending on the referred LU. Only the operations that require local (at Communication System level) execution and for which data transfer has not yet begun, can be cancelled.

The CANCEL macro instruction can be issued only when the asynchronous handling has been specified (ASY option code in effect), see Figure 38.

Name	Operation	Operands
label	CANCEL	RPL= RPL address

Figure 38 CANCEL macro instruction

RPL= RPL Address: indicates the address of the RPL associated with the request that is to be cancelled.

EXECRPL

The EXECRPL macro instruction permits the Application program to re-issue a previously issued request without modifying the related RPL.

The EXECRPL macro instruction can be used to execute any RPL based request except CHECK, CANCEL or another EXECRPL macro, see Figure 39.

Name	Operation	Operands
label	EXECRPL	RPL= RPL address

Figure 39 EXECRPL macro instruction

RPL= RPL Address: indicates the address of the RPL to be executed.

TESTLC

The TESTLC macro intruction enables the Application program to test and to get information on the status of its LU. or, if a session has been established, on the status of the logical path connecting the LUs in session, see Figure 40.

Name	Operation	Operands
label	TESTLC	RPL= RPL address %,LU= ^+LOCAL+^REMOTE^PATH

Figure 40 TESTLC macro instruction

RPL= RPL Address: indicates the location of the RPL that describes the TESTLC operation

LU= LOCAL!REMOTE!PATH: indicates the LU to be tested:

LOCAL: Only the local LU has to return its logical status

REMOTE: only the connected LU (in a session) has to return its logical status.

PATH: the full logical path status has to be returned.

INQUIRE

The INQUIRE macro instruction enables the Application program to enquiry about the Network environment, using the services offered by the Communication System. This macro has not been implemented in RPCNET, until now.

MAIL

The MAIL macro instruction enables the Application program to send and receive messages to and from another Network user or Network operator, utilyzing the mailing service offered by the RPCNET Communication System.

The MAIL macro instruction can be used only referred to an already opened LU. When used to receive messages, the MAIL macro instruction sets up a "mail box" that remains active until a mail message arrives, closing the mail request operation. When used to send a message, the MAIL macro instruction enables the Application program to specify the addressed identifier and the message text. The Communication System, on RNAM request, prepares the mail message, and forwards it to the appropriate destination. The mail request operation is considered closed as soon as the message BIU leaves the node. The information carried by the mail message as text can not be more than 128 bytes long, see Figure 41.

Name	Operation	Operands
label	MAIL	RPL= RPL address %,RPL field name= new value %,... %,MAILOPT= !SEND!RECEIVE %,BOXLAB= mail box identifier

Figure 41 MAIL macro instruction

RPL= RPL Address: indicates the location of the RPL that describes the MAIL operation.

MAILOPT= Option Code: indicates options that are to affect the mail request.

SEND!RECEIVE: indicates whether the Application program wants to send a mail message (SEND), or to set-up a mail box (RECEIVE).

RPL Field Name= New Value: although any RPL operand can be specified, the following operands apply to the MAIL macro instruction:

AREA= Data Area Address: the area at the location indicated by this field is to be used to extract the message text or to place the incoming mail message.

AREALEN= Data Area Length: indicates the length (in bytes) of the data to be transferred.

LOCID= Location Name: this field is used to indicate the Network location where the message is to be sent. The LOCID operand contains either the address of the identifier, or the identifier itself.

APPLID= this field indicates the name of the Application which the mail message is addressed to. The APPLID operand contains either the address of the identifier, or the identifier itself.

5.3.4 Macro instruction return codes

RNAM posts return code information in a general purpose register and for the RPL based macros, in certain fields of the requested RPL.

These fields are referred to as "feedback" fields. The manner in which the general register and the feedback fields are posted depends on whether synchronous request handling, asynchronous request handling with a ECB, or asynchronous request handling with an RPL exit-routine is used. RNAM always sets the general register to zero if a request has been accepted or has completed normally. When a request is not accepted, or it is completed abnormally, RNAM schedules the LERAD or ERAD exit-routine. If LERAD or ERAD exit routine is executed, upon return of control to the next sequential instruction, the general register contains whatever value was placed in it by the exit-routine. If RNAM cannot find an exit to schedule, then it sets the general register, and returns control to the next sequential instruction. RNAM, at now, uses only one non-zero return code in the general register: 4. This is termed a general return code. The errors are organized in 6 classes, according to the program recovery action that is appropriate for each error. To summarize:

- There are 2 general return codes: Ø (normal), 4 (abnormal).
- There are 6 recovery action codes that apply for abnormal completion. These are posted in the RTNCD field of the RPL. If LERAD or ERAD are invoked, the exit-routine can return its own general register value to the next sequential instruction.
- There are numerous specific error return codes, that apply for abnormal completion. These are posted in the FDBK field of the RPL.

5.3.3.1 Specific error return code (FDBK): The return code set in the FDBK field is meaningful only when it is considered together with the recovery action return code in the RTNCD field. The specific error return codes apply only when RTNCD contains a non-zero recovery action code.

RTNCD = 4 Special condition

FDBK:1 Cancel issued with I/O in progress. A CANCEL macro instruction has been completed abnormally, because data transfer is in progress No effect on the referred Application.
2 Cancel issued successfully. The operation has been terminated due to a CANCEL request.
3 Unknown Network destination specified. The request addresses a destination unknown to the Network.
4 Required binding message not sent. The BIND message is abnormally terminated, due to the absence of the binding message.
5 Password incorrect. The authorization keyword is incorrect.
6 BIND refused by the requested Application. The requested Application, after checking the bind message, has refused the connection.
7 Referred LU not in session. An in session request has been made to a LU not in session.
8 LU not in "send" state. A SEND request has been made to a LU not in "receive" state.
9 LU not in "receive" state. A RECEIVE request has been made to a LU not in "receive" state.
1Ø Inquiry incorrect. An incorrect INQUIRY request has been made (reserved for future use).
11 Inquiry information not available. The required information is not available (reserved for future use).

RTNCD = 8 Retry

FDBK 1 Temporary storage shortage. RNAM is temporary unable to receive enough storage to process the request. The request can be re-issued (for ex. via a EXECRPL).
2 No LU available. The LU request has not been accepted due to a temporary unavailability of the Communication system to assign a LU. The request can be re-issued.
3 No Application port available. The BIND request has been refused due to a temporary unavailability of a port (LU) on the requested Application. The request can be re-issued.

RTNCD = 12 Data integrity damaged

FDBK 1 Input area too small. The issued input request specified an input area that is too small RNAM has placed the required length (in bytes) in the RPL s RECLEN field. Only a part of the data has been placed in the area.
2 Incoming response indicates exception condition. A negative response to the send data request has been received.
3 Exception condition for incoming data. Data has been received for which an exception condition exists.

RTNCD = 16 Environment error.

FDBK 1 Session partner unreacheable. The Communication System is unable to reach the other session partner. The session can be considered closed or suspended.
2 Network in shut-down. The referred Network interface has decided to shut-down. All the Application could close its sessions and release all LUs.
3 Requested Application not available. The requested Application (for bind) is not available at the indicated Network location.

RTNCD = 20 Logical Error

FDBK 1 Control block invalid. The RPL's LUCB field does not contain the address of a valid LUCB.
2 Zero EXIT field The RPL indicates that the ECB-EXIT field is being used as an EXIT field, but the RPL Exit routine address is zero. No RPL Exit routine has been scheduled.
3 Zero ECB field. The RPL indicates that the ECB-EXIT field is being used to point to an external ECB, but the address in the related field is zero. No ECB has been posted.
4 Inactive RPL checked. CHECK was issued for an inactive RPL (an RPL that had been posted complete, and for which CHECK was issued successfully). All RPL based macros, however, must use an active RPL; an RPL cannot be checked twice.
5 Active RPL referred. A request has been made for an active RPL (a RPL that has an operation pending). All RPL based macros, except CHECK, must use an active RPL.
6 CANCEL issued for an inactive RPL. A CANCEL request has been issued for an inactive RPL (a RPL that had been posted complete, and

for which CHECK has been issued successfully.
7 Invalid option. The request failed because of an incorrect setting of the involved option.
8 Invalid data area. Either all or part of the output area lies beyond the addressable range of the Application program.
9 Invalid data or length. For an input operation either an input area address beyond the addressable range of the Application program has been supplied, or the area length has been invalidly indicated as zero
10 Max. number of ports for the Application exceeded. With this OPENLU request, the specified max. number of ports for the Application has been exceeded.
11 Invalid destination identifier. The identifier for the Network location to be addressed has been invalidly specified.
12 Invalid Application identifier address. The identifier for the Application to be addressed has been invalidly specified.
13 Invalid password or password address. The authorization keyword to be utilized in addressing the Application has been invalidly specified.
14 Application identifier unacceptable. The specified Application identifier is already active or not valid.
15 Invalid BIND or INVITE request. The specified request is unacceptable because the referred LU has Network requests (MAIL or INQUIRY) pending.
16 Network service request unacceptable. The specified Network service request (MAIL or INQUIRY) is unacceptable because the referred LU has a session active.

5.4 User Application Protocols

5.4.1 Introduction

The introduction in the RPCNET of a general access method like RNAM, makes it possible to write programs (in Assembler language), which can communicate each other using the Communication Systems.

These programs, in the RPCNET, were called Applications: every user can write an Application, but there exist some special Applications. that we could call System Applications, which have to perform more general functions and/or services.

These Applications must have their own, well defined protocol, and very often they require an operating system modification.

In RPCNET, two System protocols were studied and implemented as System Applications:

a) the Virtual Terminal Protocol

b) the File Transfer Protocol.

5.4.2 File Transfer Protocol

One can consider a File Transfer in two different ways:

- in the first way files are transmitted from a "spool system" to another one, without any other intervention of the user than the first command (for ex. SEND).

- in the second way, files are considered statically stored in a remote device (tape or disk), and they can be accessed as they were in a local device.

The first kind of File Transfer was called Spool-to-Spool Transfer Protol, while the second one is called Remote Files Access Protocol. In the RPCNET, the Spool-to-Spool was designed and implemented, while the Remote File Access Protocol is in course of final definition, and was implemented only in one particular environment, the VM/370 environment. So that, we will describe in the following only the Spool-to-Spool Protocol.

5.4.2.1 The Spool-to-Spool Protocol. Some of the problems to be solved for such a protocol were:

- Type of Logical Channel to be used

- Rules for opening and closing communication

- Coded information describing files (File Tag)

- User Data Unit (UDU) in a Spool-to-Spool communication

- Acknowledgment and error recovery

Since a spool file exchange is typically a one-way communication, simplex Logical Channels are used.

A Channel is opened each time a Spool System (Sender) has to send a spool file (or a set of spool files) to another Spool System (Receiver). The Channel is released when the transmission has been successfully completed. This Channel is the normal vehicle for transmission of File Tags and data.

Other kinds of information, however, need to be exchanged between Sender and Receiver. These are: connection requests (the channel does not yet exists; messages of acknowledgment and requests for error recovery. These latter

must be sent in the opposite direction to the main flow.

For connecting purposes, RPCNET offers the "BIND message" facility, which allows the Sender to send also a little amount of information to its Receiver. The RPCNET BREAK will be used for messages which must go from Receiver to Sender. The underlaying hypothesis is that Break packets cannot get lost. In other words, the subnetwork is supposed to guarantee the arrival of this type of messages.

As a very first approach, card image and print lines seem to be the natural transmission units for spool files. It must be considered, however, that the percentage of trailing blanks in spool lines or cards is generally very high. Considering also that the physical unit of transmission (packet) in the subnetwork may be much longer than lines or cards, the disadvantages in performance that such a choice could bring is easily seen:

- Logical Channel overhead: whatever amount of data is contained in a packet, it needs a fixed length amount of protocol information. This implies that the percent of data information being transferred on a Logical Channel could be very low;

- System overhead: checking protocol must be done on every packet, independently on the amount of useful data it may contain;

- resource overhead: along all the path from Sender to Receiver, there could be an overallocation of buffers in the intermediate Nodes.

Some of these disadvantages could be eliminated by packing printer lines or cards into packets. However, this solution would have a non-negligible CPU time cost, and would be dependent on the packet length. So, in order to mantain independence from subnet packet length, and in order to minimize the percentage of control information, we have to use spool disk blocks as User Data Units (UDUs) or BIUs, in RPCNET notation.

A spool block however, cannot be considered as a standard unit. In fact, the various operating systems which can be present in a Network, will generally have different spool systems. This implies, in particular, different lengths and different logical structures (formats) of blocks. As far as format is concerned, a "network spool format" was to be decided upon, which is unique in the Network. Every spool system shall be made able to send and

receive files using this standard format.

However, in each particular spool-to-spool activity, Spool Systems have been given the possibility to choose some other format that seems to be more suitable for that particular activity. This will happen of course, when both Sender and Receiver run under the same operating system; but it can be useful, also in other cases, that only one of the two partners adapts to the other's format, instead that both of them adapt to the standard net format.

Standard net format is shown in Figure 42. In the last block of the file, the end-of-block character FF is replaced by the end-of-file character EF.

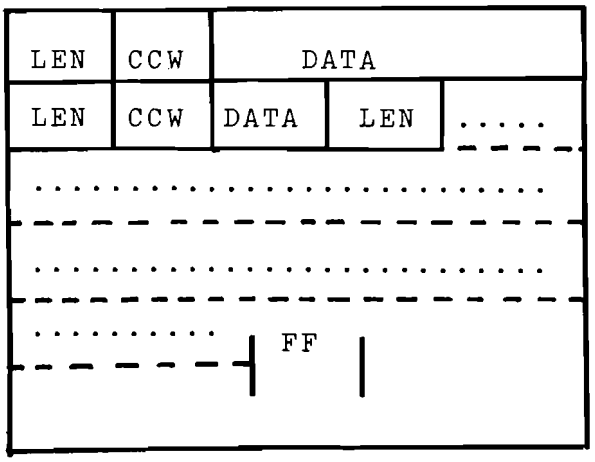


Figure 42 NET format for spool blocks

As far as output files are concerned, some information is necessary at the receiving station, concerning name, type and birthdate of files, and, possibly, about the output device to which they must be addressed. All these parameters (File Tag) are grouped at the beginning of the first BIU (block), as shown in Figure 43. At present time, the File Tag area is left empty for input files. For batch-oriented Spool Systems, FILENAME will mean jobname, and FILETYPE can be replaced by programmer's name. OUTDEVID denotes a particular remote or local output device.

The size of the BIUs is decided while establishing the Logical Channel, according to the rule that the system having the larger block size shall adapt to the "smaller" one. In fact, it is more difficult to handle core buffers larger than those normally used in the System, than sending

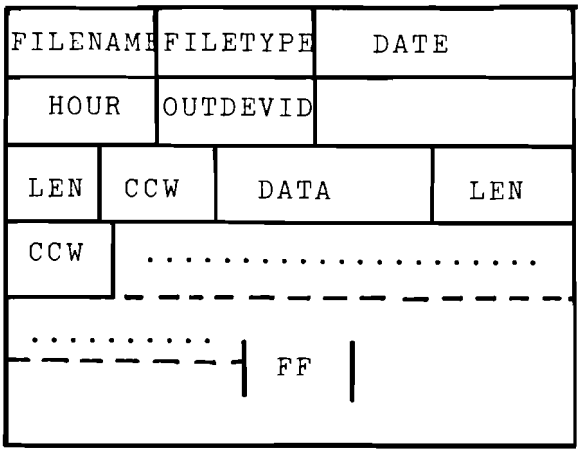


Figure 43 BIU format for File Transfer

or receiving a block splitted in two or more BIUs.

A session for opening a communication is made up of three steps: INVITE, BIND and BIND RESPONSE.

Since no information can be received if it has not been requested, the Receiver must notify to the Network its availability to perform Network operations. This is done by means of the INVITE function, that is, a request to the local Communication System to receive a message from the Network. Information supplied by INVITE contains:

- Receiver identification (SPOOL)
- Address of a fixed size core area to store a message from a partner asking to open a communication.

The Sender spool system asks to open the communication by sending a BIND to its partner. BIND contains the following information:

- Type of file(s) to be sent (input, punch print);
- Spool block length;
- Spool block format(s).

In order to use the BIND, the user must supply the following information:

- Destination Node;
- Destination Host;
- Destination Application Name (SPOOL).

It is assumed, in any case, that a BIND must receive an answer, which specifies the status of the Receiver. This status may be one of the following:

- Ready and allocated;
- Busy;
- Not active (not present at the moment in the Network);
- Invalid destination addressing.

In the first case, a communication can be established; in the last case, the operator or the originating process will be notified. In the intermediate cases, one of the following procedures can be used:

- Retry later;
- Retry after an operator command;
- Retry immediately;
- Retry when another file has to be sent to the same destination;
- Retry when a Network re-configuration occurs;
- Etc.

According to its own parameters and to those of its partner, the Receiver decides length and format of the BIU to be used. After that, it sends back a BIND RESPONSE to the Sender; this is an acceptance of connection and contains the following information:

- BIU length to be used;
- BIU format to be used. According to the mentioned before hypothesis of BREAK facility, no error recovery procedure will be implemented concerning the BIND RESPONSE.

It can happen that a block sent by the Sender does not reach the Receiver, and gets lost. This happens, for

example, when a packet gets lost because a Node has acknowledged the packet, and fails before sending it.

In order to guarantee to the receiving process the completeness of the file, both Sender and Receiver need to be involved. The Receiver should be able of recognizing the lack of blocks, the Sender of sending them again, and the Receiver of reconstructing block sequence within the file.

First of all, each block needs a sequence identification number, which must be generated by the Sender. This number will be used both by the Receiver, for check and response, and by the Sender, for any possible retransmission. The Sender has to keep pointers to the blocks just sent until it is made aware that a certain number of them is successfully arrived. Otherwise, it should read again the whole spool file, looking for those particular blocks.

Therefore, at the Sender side, this protocol requires a table of pointers to the blocks. The size of the table is to be decided during the connection phase, with an algorithm which will be discussed later. Each entry is composed of:

- the sequence (or identification) number of the BIU;

- the complete disk address which allows the Sender to resend the BIU when necessary.

At the Receiver side, the sequence of received blocks is controlled before writing them on disk. When the normal sequence is broken because of the absence of one or more blocks, this fact is registered in a "hole table". This table will be used both for asking the Sender to resend lacking blocks, and for writing these blocks in the proper area of disk when they finally arrive. In fact, space is reserved in every case in the receiving buffer or on disk, and the receiving process goes on. Each element of the table contains complete information about every missing block, that is to say, each entry is composed of:

- sequence identification number of the missing block;

- complete address of the disk area reserved for the block

- pointer to the next block on disk, which will be written with the block for chaining purposes.

When the Receiver realizes that the normal sequence is interrupted by one block whose sequence identification number is less than it was supposed to be, a scan of the "hole table" is made. If that sequence identification number is registered on the table, the block is written in the reserved disk area, and the respective entry is erased from the table. If, on the contrary, the sequence identification number is not in the table, the block is considered as a duplicate, and for this reason ignored.

The "hole table" has the same number of entries as the corresponding Sender table. The size of the table is choosen in such a way that one recovery is possible before the space in the table is exhausted. That is to say that failures, as a rule, are supposed to be recovered at the first attempt.

Let 2N be the number of entries in the table. Whenever a received block has a sequence identification number equal to (or greater than) N, 2N, 3N ..., the Receiver sends a message to its Sender for signalling the state of the previous N blocks. If all N blocks have been received properly, the Sender releases the corresponding half table In case of loss of one or more of these blocks, the Sender stops normal sending sequence for resending the missing blocks.

The integer N must be chosen in such a way to allow the arrival of an acknowledgment about the N blocks pointed in the first half of the table, before the second half has been completely filled in. In order to comply with this requirement, it is sufficient that the shipment time of N BIUs is greater than, or equal to the time requested to repeat twice the following sequence:

- one BIU from Sender to Receiver;

- one control block from Receiver to Sender.

Let be:

2N number of blocks in buffer

La number of characters per control block

Lb number of characters per BIU

S line speed in characters per second

n number of links between Sender and Receiver.

Then, the time requested for sending N blocks will be:

$$T = N * Lb/S$$

and the time requested for repeating twice the sequence above described is:

$$T' = 2n * La/S + 2n * Lb/S$$

We want to find N in such a way that T>= T'. That is:

$$N>=2n*(1+ La/Lb)$$

As La is supposed to be not greater than Lb, N=4n seems to be a good solution for our purposes.

The number n of links between Sender and Receiver can be obtained during the Connecting phase.

It may happen that the Sender fills up one half of its table before the other half can be released. This happens in two cases:

- The Sender has sent 2N BIUs without any feedback.
- Some resent blocks have been lost again.

In the first case, the Sender starts a timeout, and if nothing arrives within a fixed time, the Sender closes the connection. In the second case, the last received holes are resent again, and a timeout mechanism is started; if the timeout expires without replies, the Sender closes the connection.

On the other side, the Receiver starts a timeout mechanism each time a block arrives; when the timeout expires, a break message is sent, containing the present situation of received blocks, and a second timeout is started. If also the second timeout expires, the connection is closed.

After a whole file has been successfully sent, the Sender can begin the transmission of another file. If there are no more files of the same type for the same destination, the Sender issues a CLOSE, that is to say, a request to close the Logical Channel used until that time. Then the Sender expects no more messages from the Network.

When a Receiver got an entire file, it expects to receive an UNBIND or another file. If nothing arrives, the Receiver uses the same procedure as in the case of interruption of communication during file transmission.

5.4.3 Virtual Terminal Protocol

The Terminal Application is made up of a line driver that maps a real terminal into an idealized terminal (here called a Virtual Terminal, or VT), a Network Command Processor, and a module for communicating with the Host Application via the RPCNET Logical Channel.

The Host Application maps the Virtual Terminal Protocol into actions meaningful to the Host computer. As far as the Host and Terminal Applications were concerned, a full-duplex facility was easier to work with. In RPCNET the Logical Channel provides a full-duplex facility (if we use the BREAK facility), with error detection in case of loss of messages, or out-of-sequence messages between the components. The error recovery is the same as for both loss and out-of-sequence.

It is worth noting that the first implementation of RPCNET was intended to insure that the message loss occurred only with a loss of a Network Node containing the message. Thus, although the error recovery is necessary, it is seldom invoked, except to prevent "hang" conditions, in the rare case of a crash of a node holding a message relevant to the Host and Terminal intercannection.

Relevant to this discussion are the SEND, RECEIVE, BREAK and TESTLC RPCNET operations. Due to the nature of the Logical Channel, it is necessary to "change direction" before a Receiver can send and a Sender can receive.

In the RPCNET VTP, the terminal side is always kept in RECEIVE mode, while the Host side is always kept in SEND mode. This produces only minor restrictions on the terminal side, which has to stop its current RECEIVE, in order to send its message by BREAK.

Other functions for controlling the Logical Channel of interest here are those used for making and breaking connections between the Applications. Typically, the Host Application will have as many INVITEs in action as there are free ports (emulated terminals) available for connection to its Host. When a user of a terminal requests a connection to a Host, a BIND is done, and if there is a corresponding INVITE at the Host, then the connection is made.

Since the "direction" of the Logical Channel is initially not as desired (the terminal is the Sender, the Host is the Receiver), the terminal Application issues a SEND with change direction, to turn the channel around. At any time, either Application can do an UNBIND to break the

connection. This is normally done either at the request of the terminal user, or when the Host computer "crashes". The Figure 44 illustrates the states in which a virtual terminal can be.

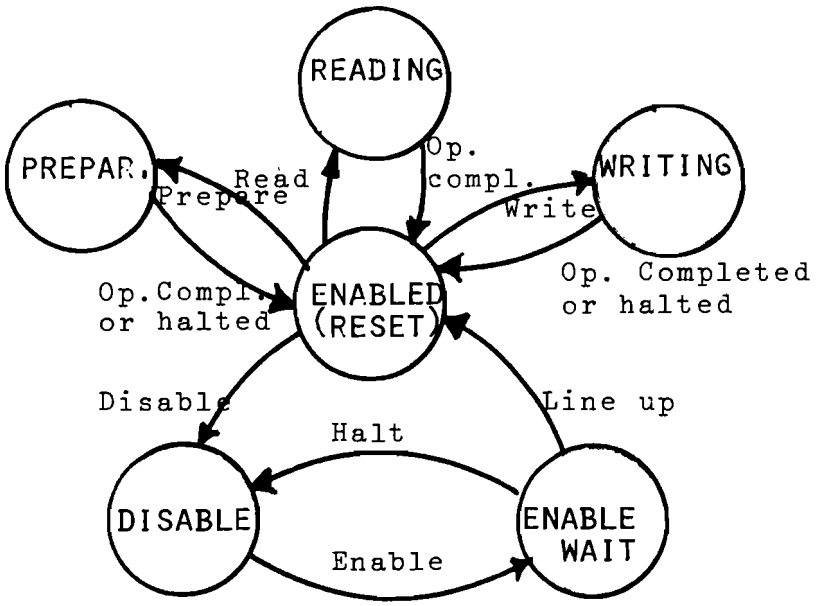


Figure 44 States of a Virtual Terminal Protocol

The write state is when the terminal is writing to the terminal user, and the read state is when the user can type input on the terminal. The prepare state is used to watch for asynchronous attentions, or the disconnection of the terminal. The prepare, the write and the read state are haltable by the program driving the VT.

The virtual terminal character set is the EBCDIC.

The behaviour of the two Applications when contacting each other is illustrated in Figure 45.

There is a state diagram for each Application, indicating how the BIND and INVITE operations are used. The BIND message used by the Terminal and Host Applications is described later. If the BIND request is accepted by the Host, a BIND message is sent by the Host Application. Finally, the terminal BIND request operation ends, with an indication of the success or failure of the connection making attempt.

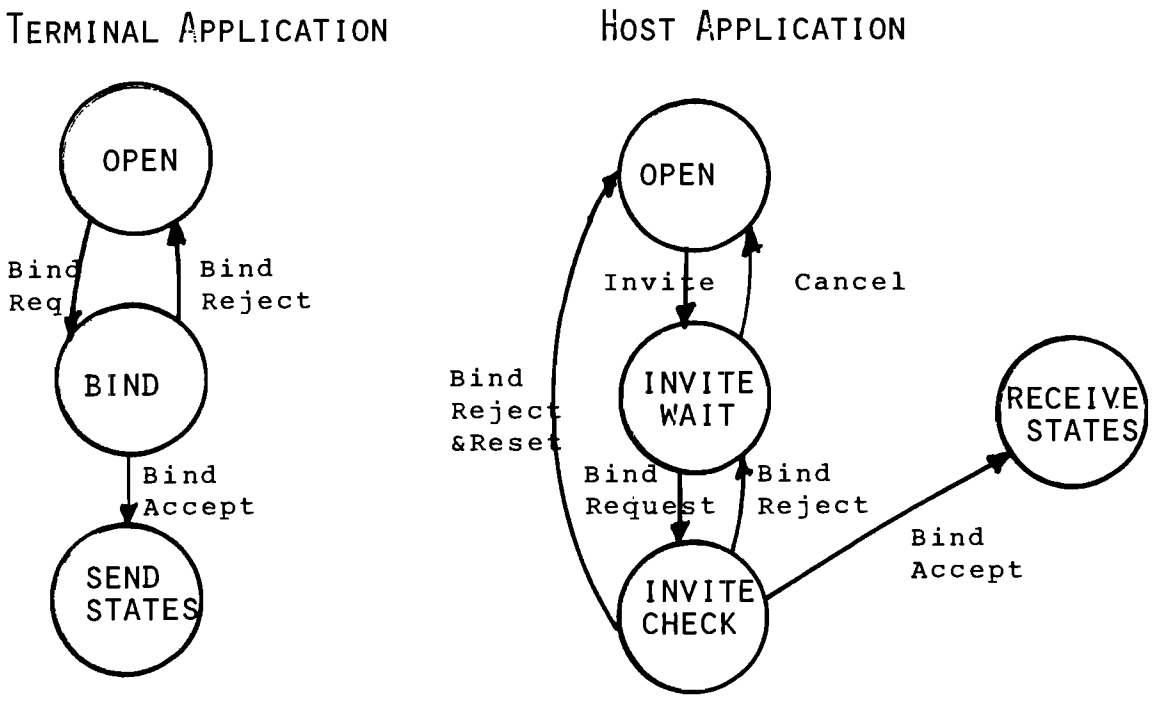


Figure 45 State diagram for the Virtual Terminal

The BIND message can be up to almost a packet in length, allowing a very long protocol message. In the RPCNET implementation, the first byte is used to specify the type of protocol to be used, i.e. the type of the Host to be connected with. At now, the hexadecimal value '40' is used to specify a VM/370 Host, '80' is a S/7, and '20' is an OS/TSO. Other values are free. According to the first byte, the length of the BIND message can be different: for example, in the VM/370 implementation, there are 5 bytes more, to describe the device type, the device class, etc.

Read, write, halt, attention and related activities proceed until one of four situations arise, signalling the end of the session. If the terminal is disconnected, or the Host goes down, or if the Host disables the terminal, the session is ended. In addition, behavior of the Network

itself can end the session, as when access is lost between the Application due to Network failure.

All the messages, during the session, are "data", with respect to RPCNET facilities that support Logical Channels These messages compose the elements of VTP. The first byte reflects the four classes of messages in bits 0 and 1. A value of 00 designates terminal and Host action messages, such as write, read, halt, attention, and acknowledgments thereof. A value of 01 designates terminal and Host oriented control messages, such as type-ahead control. A value of 10 designates a Network oriented action message, such as the "do TESTLC" message. A value of 11 designates Network oriented control messages, such as disable, host down, or terminal down.

Bit 2 of the first byte indicates which Application is the Sender. A value of 0 designates the Host, and 1 designates the terminal. Bits 3-7 are used to further distinguish the message.

byte 1	byte 2	byte 3	
00000000	n------n		halt n
00000001	n------n		read request n
00000010	n------n	data....	write request n
00000011	m------m		acknowledge attention m
00100000	n------n		acknowledge wr. or ht n
00100001	n------n	data....	read reply n
00100010	n------n	m------m	wr. n ended by attn. m
00100011	m------m		attention m
01100000	XY00CCCC		wr. ahead mode control

XY=00 no write ahead; no message save
XY=01 no write ahead; message saved for error recovery
XY=10 write ahead allowed
CCCC= count of messages the Host Application can write ahead.

10100000		do TESTLC
11000000		disable
11100000		terminal disconnected

The Host Application does not need a queue for messages. It needs only a counter to record the number of unprocessed attentions, and a variable to hold the latest attention sequence number. The terminal Application will need queues for both input and output, to communicate with the Host Application. The output queue is to hold items delayed because the current RECEIVE could not be cancelled since it was busy receiving a message. The input queue holds read write and halt requests from the Host. There is a second queue between the terminal and the input queue; it is used for processing terminal Application control messages, and it is otherwise empty.

The Host Application assigns numbers sequentially and cyclically running from Ø,1,...,255,Ø,.. to read, write and certain halt requests of the host computer.

A halt is assigned a number if it halts a read or write, otherwise it is not counted and is ignored. Notice of each such halt, read or write is sent to the terminal Application, where each is enqueued for presentation to the terminal handler

As each is processed by the terminal handler, it is acknowledged by a message sent back to the Host Application. The message contains the message number assigned by the Host Application originally. The acknowledgment mechanism is necessary to prevent the Host from flooding the Network with a series of write messages. The acknowledgments messages are of several sorts: normal write or halt acknowledgments, write ended with attention, and read reply (which includes the data that was requested to be read from the terminal).

Successive attentions are designated in the message formats by successive attention numbers, running cyclically from Ø,1,...,255,Ø,... The terminal Application assigns the sequence numbers for attention.

When an attention is presented to the Host computer, by the Host Application, a message acknowledging the attention is returned to the terminal application.

Two messages designate an attention, one normal (not interrupting a read or write), and one ending a write. Attentions ending a read are not included in the attention mechanism described here, since by convention they are designated by a read reply message, that lacks a "new line" character as its last character. One message is used to designate that an attention has been presented to the Host. Such presentation is done by ending the current (if one) or the next read, write or prepare with an appopriate attention indication. No more than two attentions can be sent from the terminal to the Host at any time which have not yet been acknowledged. Note that attentions ending a read are not counted in this figure.

APPENDIX A

Y Cable Hardware Connections

Y Cable Hardware Connections

CABLE ADAPTOR: CA1A

2705 BSC TSTM.
female connection

protective ground	1----
transmit data	2----
receive data	3----
request to send	4----
clear to send	4----
data set ready	6----
signal ground	7----
data carrier detect.	8----
transmit timing	15---
receive timing	17---
data terminal ready	20---

----1	protective ground
----2	transmit data
----3	receive data
----4	request to send
----5	clear to send
----6	data set ready
----7	signal ground
----8	data carrier detector
----15	transmit timing
----17	receive timing
----20	data terminal ready

2703 BSC RCV.
female connection

protective ground	1----
transmit data	2----
receive data	3----
request to send	4----
clear to send	5----
data set ready	6----
signal ground	7----
data carrier detect.	8----
transmit timing	15---
receive timing	17---
data terminal ready	20---

APPENDIX B

An Example of the NETCHANGE Protocol

An Example of the NETCHANGE Protocol

In this Appendix B we give an example of how the protocol works for a particular network.
The network is the one described in Figure 2.

Initial configuration

A	AB	AD		
A	-	-	-	-
B	1	3	1	B
C	2	2	2	B
D	3	1	1	D
E	3	2	2	D

B	BA	BC		
A	1	3	1	A
B	-	-	-	-
C	3	1	1	C
D	2	2	2	A
E	3	2	2	C

C	CB	CD	CE		
A	2	2	3	2	B
B	1	3	3	1	B
C	-	-	-	-	-
D	3	1	2	1	D
E	3	2	1	1	E

D	DA	DC	DE		
A	1	3	3	1	A
B	2	2	3	2	A
C	3	1	2	1	C
D	-	-	-	-	-
E	3	2	1	1	E

E	EC	ED		
A	3	2	2	D
B	2	3	2	C
C	1	2	1	C
D	2	1	1	D
E	-	-	-	-

The link EC goes down. Changes occur in nodes C and E.

A	AB	AD		
A	-	-	-	-
B				
C				
D				
E				

B	BA	BC		
A				
B	-	-	-	-
C				
D				
E				

C	CB	CD	CE		
A	2	2	5	2	B
B	1	3	5	1	B
C	-	-	-	-	-
D	3	1	5	1	D
E	3	2	5	2	D

D	DA	DC	DE		
A					
B					
C					
D	-	-	-	-	-
E					

E	EC	ED		
A	5	2	2	D
B	5	3	3	D
C	5	2	2	D
D	5	1	1	D
E	-	-	-	-

After the changes, the node C sends its Routing Table (distance column) to B and D; the node E sends its Table to D; the Tables are changed.

A	AB	AD		
A	-	-	-	-
B				
C				
D				
E				

B	BA	BC		
A	1	3	1	A
B	-	-	-	-
C	3	1	1	C
D	2	2	2	A
E	3	3	3	A

C	CB	CD	CE		
A					
B					
C	-	-	-	-	-
D					
E					

D	DA	DC	DE		
A	1	3	3	1	A
B	2	2	4	2	A
C	3	1	3	1	C
D	-	-	-	-	-
E	3	3	1	1	E

E	EC	ED		
A				
B				
C				
D				
E	-	-	-	-

The Routing Table for the node D has not been changed, so that only B sends its Routing Table to C and A.

A	AB	AD		
A	-	-	-	-
B	1	3	1	B
C	2	2	2	C
D	3	1	1	D
E	4	2	2	D

B	BA	BC		
A				
B	-	-	-	-
C				
D				
E				

C	CB	CD	CE		
A	2	2	5	2	B
B	1	3	5	1	B
C	-	-	-	-	-
D	3	1	5	1	D
E	4	2	5	2	D

D	DA	DC	DE		
A					
B					
C					
D	-	-	-	-	-
E					

E	EC	ED		
A				
B				
C				
D				
E	-	-	-	-

As the Routing Tables of A and C have not been changed, the process ends. Now, let's see what happens if the link EC comes up again.
Changes occur in C and E.

A	AB	AD		
A	-	-	-	-
B				
C				
D				
E				

B	BA	BC		
A				
B	-	-	-	-
C				
D				
E				

C	CB	CD	CE		
A	2	2	5	2	B
B	1	3	5	1	B
C	-	-	-	-	-
D	3	1	5	1	D
E	4	2	1	1	E

D	DA	DC	DE		
A					
B					
C					
D	-	-	-	-	-
E					

E	EC	ED		
A	5	2	2	D
B	5	3	3	D
C	1	2	1	C
D	5	1	1	D
E	-	-	-	-

C sends its Routing Table to B, D and E. E sends its Routing Table to C and D.

A	AB	AD		
A	-	-	-	-
B				
C				
D				
E				

B	BA	BC		
A	1	3	1	A
B	-	-	-	-
C	3	1	1	C
D	2	2	2	A
E	3	2	2	C

C	CB	CD	CE		
A	2	2	3	2	B
B	1	3	4	1	B
C	-	-	-	-	-
D	3	1	2	1	D
E	4	2	1	1	E

D	DA	DC	DE		
A	1	3	3	1	A
B	2	2	4	2	A
C	3	1	2	1	C
D	-	-	-	-	-
E	3	2	1	1	E

E	EC	ED		
A	3	2	2	D
B	2	3	2	C
C	1	2	1	C
D	2	1	1	D
E	-	-	-	-

E, D and B modify their Routing Tables, which are sent to the neighbors.

A	AB	AD		
A	-	-	-	-
B	1	3	1	B
C	2	2	2	B
D	3	1	1	D
E	3	2	2	D

B	BA	BC		
A				
B	-	-	-	-
C				
D				
E				

C	CB	CD	CE		
A	2	2	3	2	B
B	1	3	3	1	B
C	-	-	-	-	-
D	3	1	2	1	D
E	3	2	1	1	E

D	DA	DC	DE		
A	1	3	3	1	A
B	2	2	3	2	A
C	3	1	2	1	C
D	-	-	-	-	-
E	3	2	1	1	E

E	EC	ED		
A	3	2	2	D
B	2	3	2	C
C	1	2	1	C
D	2	1	1	D
E	-	-	-	-

None of the Routing Tables has been modified. The process ends, and the configuration is the same as the beginning of our example.

APPENDIX C

Protocol Packets Formats

Protocol Packets Formats

As we saw, for every Network request, there is a packet with a well specified packet format. Here we will describe the formats of all the packets for all the requests.

First of all, we must specify the codes describing the operations (note: from now on, all the codes are in exadecimal format).

X 40' BIND wait

X 41 Bad Password

X'48 BIND response

X 80' INVITE general broadcast

X'40' INVITE single broadcast

X'80' MAIL send

X'00' SEND

X'10 RECEIVE

X'20' BREAK

X'30' TESTLC

X'50' CANCEL

X'58' REJECT

X'70' UNBIND

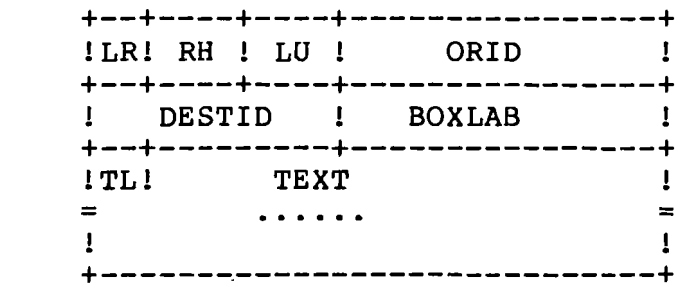
X'B0' INQUIRE

X'C0' OPEN

X'F0' CLOSE

In the following, there are the messages exchanged by the Interface Functions Layer components.

MAIL



where:

LR Last packet received in sequence (1 byte)

RH Request Header (code of the request) (2 bytes)

LU LU Identificator (2 bytes)

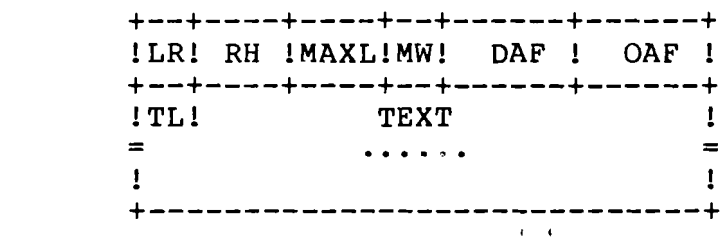
ORID Originator identification (8 bytes)

DESTID Destination identification (8 bytes)

BOXLAB Boxlabel name (8 bytes)

TL Text length.

BIND RESPONSE



where:

LR Last packet received in sequence (1 byte)

RH Request Header (operation code) (2 bytes)

MAXL Maximum BIU length (2 bytes)

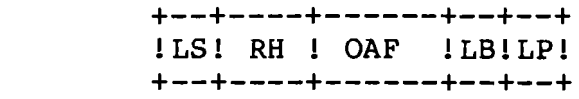
MAXW Maximum window width (1 byte)

DAF Destination Address Field (3 bytes)

OAF Originator Address Field (3 bytes)

TL Text length (1 byte)

UNBIND



where:

LS Last packet received in sequence (1 byte)

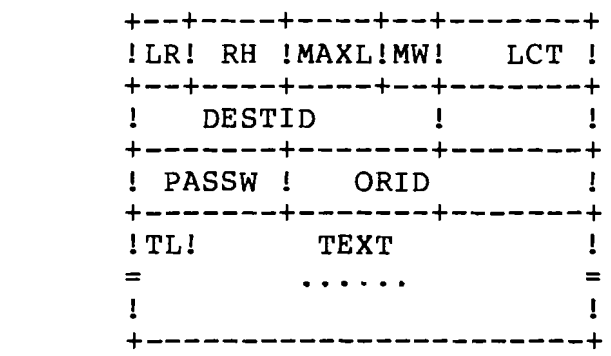
RH Request Header (operation code) (2 bytes)

OAF Originator Address Field (3 bytes)

LB Last BIU sent (1 byte)

LP Last PIU sent (1 byte)

BIND



where:

LR Last packet Received in sequence (1 byte)

RH Request Header (operation code) (2 bytes)

MAXL Maximum BIU length (2 bytes)

MW Window width (1 byte)

LCT Logical Channel Termination id. (3 bytes)

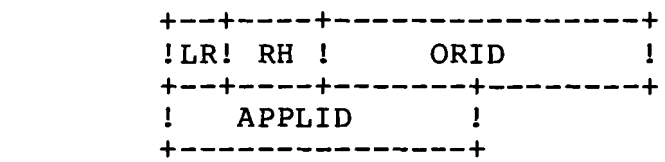
DESTID Destination identification (8 bytes)

PASSW Authorization keyword (8 bytes)

ORID Originator identification (8 bytes)

TL BIND message text length (1 byte)

INVITE



where:

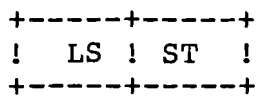
LR Last packet received in sequence (1 byte)

RH Request Header (operation code) (2 bytes)

ORID Originator id. (8 bytes)

APPLID Application id. (8 bytes)

STATUS (NSM)

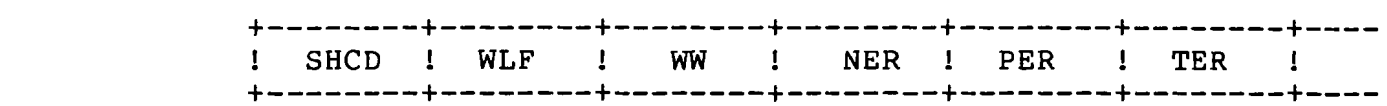


where:

LS Last packet received in sequence (1 byte)

ST NSM Restart Number (1 byte)

STATUS (SH)



where:

SHCD Session Handler Status Message Code:
X'80' Status message
X'40' Request For status
X'20' Information Message
X'01' Reconfiguration Flag

If the message is not a status message (X'80'), only three fields are used:

SHCD With the meaning described above

WLF Last PIU sent

WW Last PIU sent before a reconfiguration.

If the message is a status message (X'80'):

SHCD Status Message (X'80')

WLF Dinamic window left edge (1 byte)

WW Dinamic window width (1 byte)

NER Number of error

PER PIU in error

TER Type of error

REFERENCES

(1) Caneschi F., Ferro E., Lenzini L., Martelli M., Menchi C., Sommani M., Tarini F. : "The Architecture and the Service Facilities provided by RPCNET - An Italian Computer Network for Educational and Research Institutions", ICCC Conference Kyoto 1978.

(2) Fusi A. Editor: "CNS/VM Computer Network Subsystem for VM/370. General Information", RPCNET internal document No. UG019-00 Pisa, May 1976.

(3) Gori G. A., Maier M.: "The OS/VS Network Station (CNS-VS) in a RPCNET node", RPCNET internal document No. UG010-01 Pisa, Jan. 1976.

(4) Guidotti P., Lazzeri L., Lenzini L., Martelli M.: "Network Manager implementation on System/7", RPCNET internal dicument No. UI023 Pisa Oct. 1975.

(5) Lazzeri L., Lenzini L., Martelli M.: " A proposal for the introduction of traffic load in NETCHANGE protocol", RPCNET internal document No. IG015-00 Pisa Jan 1975.

(6) Springer A., Lazzeri L., Lenzini L.: "The implementation of RPCNET on a minicomputer", Computer Communication Review, Cambridge Ma., Jan 1978.

(7) Tajibnapis W. D.:" The design of a Topology Information Maintenance Scheme for a Distributed Computer Network , ACM Conference, S. Diego Ca., Nov. 1974.