DECnet Digital Network Architecture
Phase IV
NSP Functional Specification
Order No. AA-X439A-TK

digital

software

DECnet Digital Network Architecture
Phase IV
NSP Functional Specification

Order No. AA-X439A-TK

December 1983

This document describes the NSP architecture, which models that part of the DECnet software that supports the creation and destruction of logical links, error control, and flow control. NSP is the protocol of the End Communications layer. The End Communications layer is part of the Digital Network Architecture.

SUPERSESSION/UPDATE INFORMATION: This is a new manual.

VERSION: 4.0.0

To order additional copies of this document, contact your local Digital Equipment Corporation Sales Office.

digital equipment corporation · maynard, massachusetts

First Printing, December 1983

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

Distributed Systems Publications typeset this manual using DIGITAL's TMS-11 Text Management System.

MGTPEALL

CONTENTS

1.0 INTRODUCTION

This document describes the structure, functions, interfaces, and messages of NSP, the protocol of the End Communications Layer. The End Communications Layer is that part of the DIGITAL Network Architecture (DNA) that models the software (or hardware) enabling the creation and destruction of logical communication links, data flow control, end-to-end error control, and the segmentation and reassembly of messages.

DIGITAL Network Architecture is the model on which DECnet implementations are based. A DECnet network is a family of software modules, data bases, and hardware components used to tie DIGITAL systems together for resource sharing, distributed computation or remote system communication.

DNA is a layered structure. Modules in each layer perform distinct functions. Modules within a single DNA layer (but typically in different computer systems) communicate using specific protocols. Modules in different layers (but typically in the same computer system) interface using subroutine calls or a system-dependent method. In this document interfaces are described in terms of calls to subroutines.

This specification describes Phase IV NSP architecture. In Phase II, an earlier version, Session Control was part of NSP. With Phase III and Phase IV, Session Control has been logically separated from NSP, and the interface between the two layers defined. The Session Control Layer is described in a separate functional specification. The Routing specification, also a part of the Phase II NSP specification, has been greatly expanded in Phase III and Phase IV and is contained in a separate Routing specification. Appendix E details the differences between Phase III and Phase IV NSP.

A glossary at the end of this document defines many NSP terms. This document assumes that the reader is familiar with computer communications and DECnet. The primary audience consists of those who implement DECnet systems; however, the document may be useful to anyone interested in the details of DECnet structure. The other current DNA functional specifications are:

DNA Data Access Protocol (DAP) Functional Specification, Version 5.6.0, Order No. AA-K177A-TK

DNA Digital Data Communications Message Protocol (DDCMP) Functional Specification, Version 4.1.0, Order No. AA-K175A-TK

DNA Ethernet Data Link Functional Specification, Version 1.0.0, Order No. AA-Y298A-TK

DNA Ethernet Node Product Architecture Specification, Version 1.0.0, Order No. AA-X440A-TK

DNA Maintenance Operations Functional Specification, Version 3.0.0, Order No. AA-X436A-TK

DNA Network Management Functional Specification, Version 4.0.0, Order No. AA-X437A-TK

DNA Routing Layer Functional Specification, Version 2.0.0, Order No. AA-X435A-TK

DNA Session Control Functional Specification, Version 1.0.0, Order No. AA-K182A-TK

The Ethernet - A Local Area Network - Data Link Layer and Physical Layer Specifications, Version 2.0, (Digital, Intel, and Xerox), Order No. AA-K759B-TK

The DECnet DIGITAL Network Architecture (Phase IV) General Description (Order No. AA-N149A-TC) provides an overview of the network architecture and an introduction to each of the functional specifications.

2.0 FUNCTIONAL DESCRIPTION

NSP performs the following functions:

1. Enables the creation and destruction of virtual channels (logical links) that can be used for sending messages within a network node and between network nodes.

2. Manages the movement of interrupt and normal data from transmit buffers to receive buffers, using flow control mechanisms.

3. Breaks up normal data messages into portions (segments) that can be transmitted individually, and reassembles these segments in their correct order after they have been transmitted.

4. Guarantees the delivery of data and control messages to a specified destination by means of an error control mechanism.

Section 2 is an overview of NSP, covering the following topics:

- Design scope (Section 2.1)

- Relation of NSP to the DIGITAL Network Architecture (Section 2.2)

- Routing characteristics (Section 2.3)

- Basic concepts (Section 2.4)

- Messages (Section 2.5)

- Major functions (Section 2.6)

- o Functional components (Section 2.7)

2.1 Design Scope

NSP satisfies these following design requirements:

1. Compatibility. NSP version 4.0 is compatible with previous versions of NSP except for those differences described in Appendix E.
2. Performance. NSP allows an implementation to perform without deadlocks while using dynamic buffer pools.
3. Promptness. NSP minimizes the delays incurred in moving data from one Session Control module to another.
4. Efficiency. NSP minimizes the communications overhead (for example, line bandwidth) consumed by the protocol.
5. Extensibility. NSP accommodates additional functions in the future, leaving earlier functions as a subset.
6. Fairness. If more than one logical link is established to the same destination at the same time, NSP assures that each provides useful communication services.
7. Elasticity. NSP allows an implementation to trade memory resources (both algorithm complexity and buffer pool sizes) for performance.

The following are not within the scope of Phase IV NSP:

1. Maximum throughput. NSP will not necessarily maximize the throughput of a logical link.
2. Uniform service. NSP will not guarantee the same average throughput and average delays over two logical links from a common source to a common destination.

2.2 Relation to DIGITAL Network Architecture

Figure 1 shows the relationship of the End Communications Layer to the DNA hierarchy. Each layer in DNA consists of functional modules and protocols.

Generally, modules use the services of the next lowest layer. In this document the service relationship is demonstrated in the way the interfaces are modeled -- as calls to subroutines. Note, however, that the Network Management Layer interfaces directly with each of the lower layers. Also, all the layers above Session Control interface directly with it. In fact, the upper three layers are sometimes referred to as the "end user."

Modules of the same type in the same layer communicate with each other to provide their services. The rules governing this communication and the messages required constitute the protocol for those modules. Messages are typically exchanged between equivalent modules in different nodes. However, equivalent modules within a single node can also exchange messages.

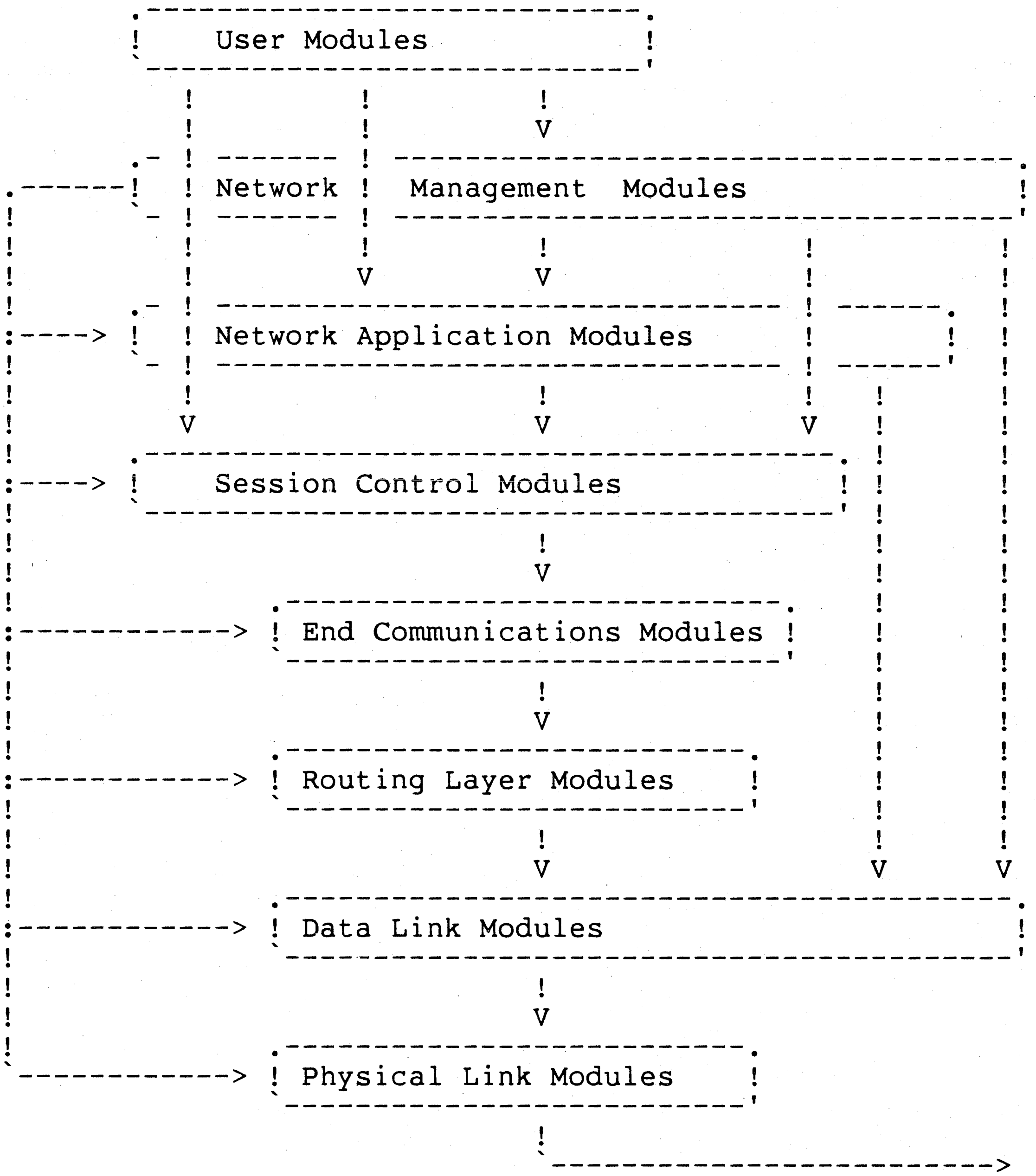


Figure 1. NSP Relation to DNA

A brief description of each layer follows in order from the highest to the lowest layer:

1. User Layer. The highest layer, the User Layer supports user services and programs. Programs such as the Network Control Program, which interfaces with the Network Management Layer, and file transfer programs, which interface with the Network Application Layer, reside in the User Layer.

2. Network Management Layer. The Network Management Layer is the only one that has direct access to each lower layer for control purposes. Modules in this layer provide user control over and access to network parameters and counters. These modules also perform up-line dumping, down-line loading, and testing functions.
3. Network Application Layer. Modules in the Network Application Layer support network functions, such as remote file access and file transfer, used by the User and Network Management Layers.

4. Session Control Layer. The Session Control defines the system-dependent aspects of logical link communication, which allows messages to be sent from one node to another in a network. Session Control functions include name to address translation, process addressing, and, in some systems, process activation and access control.

5. End Communications Layer. The End Communications Layer defines the system-independent aspects of logical link communication.

6. Routing Layer. Modules in the Routing Layer route messages, called packets, between source and destination nodes.

7. Data Link Layer. The Data Link Layer defines the protocol concerning data integrity and physical channel management.

8. Physical Link Layer. The Physical Link Layer encompasses a part of the device driver for each communications device plus the communications hardware itself. The hardware includes interface devices, modems, and the communication lines.

2.3 Routing Characteristics

NSP interfaces directly with the Routing Layer for its services. Routing is a datagram delivery service to NSP. A datagram is a block of data sent intact from one DECnet node to another. Routing sends datagrams in packets. NSP expects Routing to have the following characteristics:

1. Routing will accept a datagram at least as large as 230 8-bit bytes.

2. There is an extremely low probability that Routing will:

   a. Duplicate a datagram

   b. Deliver a datagram to the wrong destination

   c. Change the data in a datagram

3. Routing may fail to deliver a datagram.

4. Routing may fail to deliver datagrams to a given destination from a given source in the order they were transmitted.

5. Datagrams delivered to a given destination from a given source undergo a variable delay while under the control of Routing. However, this maximum delay is bounded. Datagrams not delivered within the maximum delay will not be delivered.

2.4 Basic NSP Concepts

This section describes concepts that are fundamental to an understanding of NSP.

1. Logical links and ports (Section 2.4.1)

2. Port and logical link states (Section 2.4.2)

3. Logical link identification (Section 2.4.3)

4. Data flow (Section 2.4.4)

2.4.1 Logical Links And Ports

- NSP provides a logical link service to Session Control. A logical link is a virtual connection between two Session Control modules, either between two nodes or within one node. The connection enables controlled communication between network nodes. A pair of Session Control modules may have more than one logical link between them. Each logical link is separate from all other logical links.

Each logical link must have a port at each end. A port is an area in memory, generally in a dedicated or shared pool, that contains control variables for managing logical links. Table 4 in Section 5.2 specifies these variables. NSP manages ports. Each node on a network has a number of available ports. In forming a logical link, one port is associated with another.

When Session Control requests a logical link or requests that a port be opened to receive an incoming connect request, NSP allocates a port if sufficient resources are available. When Session Control requests that a port be closed, NSP deallocates the resources associated with the port. Deallocation usually occurs after Session Control requests a logical link disconnection.

NSP also maintains a "confidence" variable in each port that has been opened. Session Control has access to this information, which is useful in detecting network failures.

2.4.2 Port And Logical Link States - Each end of a logical link is in one of a set of states at any time. In other words, each port has a state.

The states at one end of the link affect the states at the other end of the link. In this document the possible link states at one end of a link are called the port states. The logical link states are the combination of possible states at both ends of the logical link.

Every logical link has its own set of logical link states.

Session Control requests and NSP messages determine the particular states and state transitions of the logical link. NSP manages these state changes, based on the particular requests and messages it receives. Section 5 details all the normal port and logical link states and state transitions.

2.4.3 Logical Link Identification - In order for two NSP modules to manage a given logical link, each NSP module must be able to identify the link. The logical link identification consists of the port addresses at each end of the link.

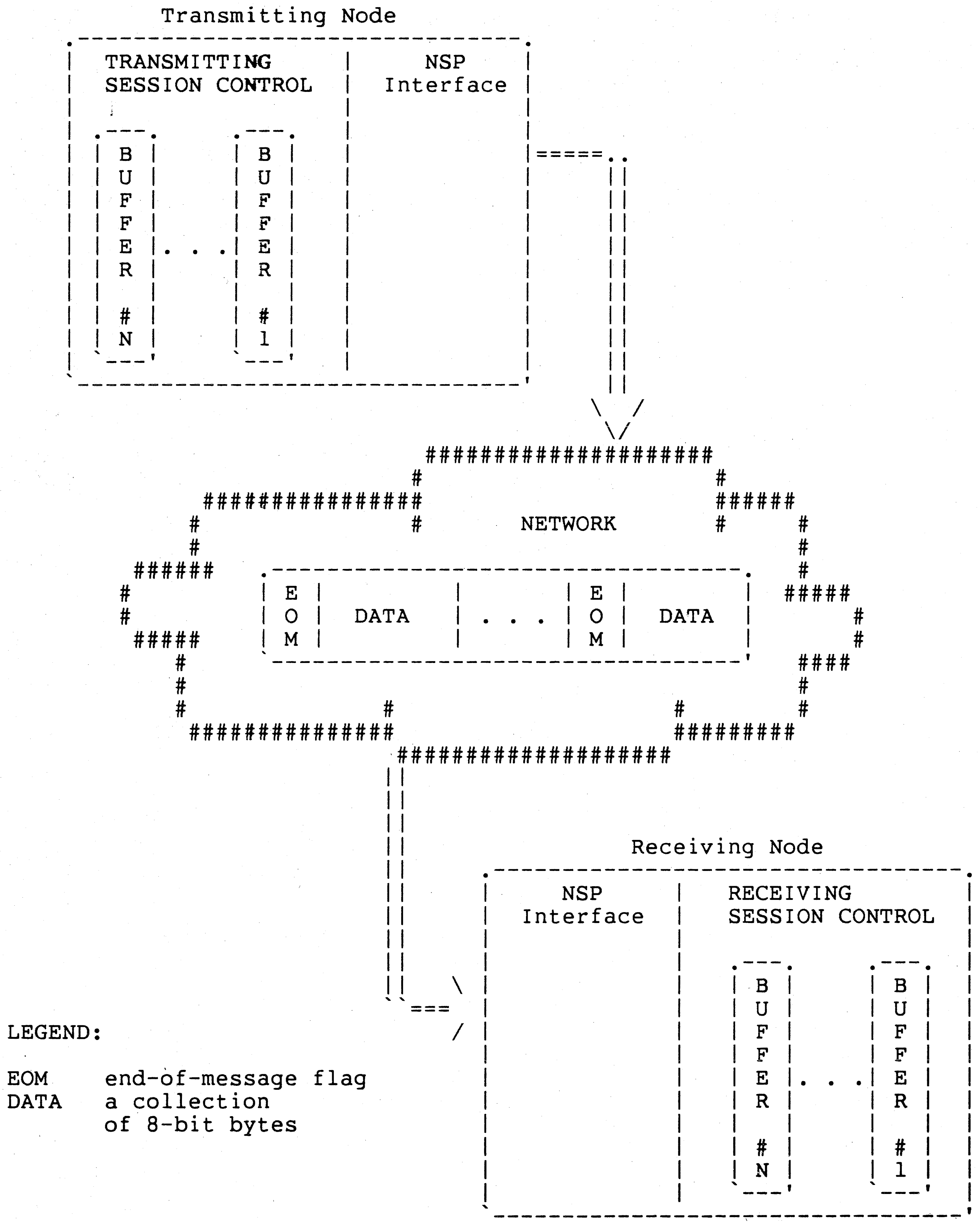
Each NSP module assigns a 16-bit numerical address to its end of a logical link. The port at one end of the link contains the address of the port at the other end of the link and vice versa. This is the way in which the two ports are associated with each other. The complete identification of the link, identifying both ends of the link, is therefore a 32-bit number.

To avoid using the same number to identify two different links, an NSP module refrains from assigning a 16-bit address it used for a previous (but now disconnected) link to a new link as long as possible. The probability that each of the two NSP modules reassigns its 16-bit address and that these two addresses are paired a second time during a connection process is extremely low. Therefore, the probability that the same 32-bit identification would exist for two different links is very low. This ensures that there will be no cross-talk between links.

2.4.4 Data Flow - After a logical link is established, data may flow in both directions (full-duplex) from transmitting Session Control transmit buffers through the network to receiving Session Control receive buffers. The size of the buffers at each end of the link is implementation-dependent. However, the data flowing through the network is always handled the same way. The NSP interface to Session Control takes Session Control data provided in DATA-XMT calls (Section 3.1). It then transforms the data to a network form. At the other end of the link the receiving NSP interface, responding to Session

Control DATA-RCV calls, transforms the data from its network form to its receive buffer form. The mechanisms NSP uses to handle data are transparent to Session Control. From Session Control's viewpoint, the data flow is as shown in Figure 2.

This figure shows Session Control data transformed from a transmitting Session Control to a transmitting NSP and then transformed back from a receiving NSP to a receiving Session Control. The NSP data does not actually move tarough the network as shown. (The DNA General Description shows how Routing packets actually move through the network.)



LEGEND:

EOM	end-of-message flag
DATA	a collection of 8-bit bytes

Figure 2. Model of Data Flow as Seen by Session Control.

The transmitting NSP appends the end-of-message (EOM) flags (Figure 2), to the data in the network form. The receiving NSP module removes these flags, places only data in the Session Control receive buffers, and then informs the receiving Session Control via a flag in a returned receive buffer whether an EOM was received. Section 3.1 details this procedure.

Throughout the data flow process, NSP preserves data order. NSP places data bytes from a single transmit buffer into the network form in the same order as they were in the buffers. NSP also guarantees that no data will be lost. Section 3.1 details the data flow procedure.

2.5 Messages

In order to provide logical link service, flow control and error control (thereby supporting the Session Control interface), NSP modules in different nodes must communicate. They do so by sending and receiving NSP messages. The NSP protocol consists of these messages and the rules governing their exchange.

There are three types of NSP messages:

- Data messages
- Acknowledgment messages
- Control messages

Table 1 summarizes the functions performed by each NSP message. Section 8 describes the message formats in detail.

Table 1
NSP Messages

Type	Message	Description
Data	Data Segment	Carries a portion of a Session Control message. (This has been passed to Session Control from higher DNA layers and Session Control has added its own control information.)
Data (also called Other-Data)	Interrupt	Carries urgent data, originating from higher DNA layers.
	Data Request	Carries data flow control information (also called Link Service message).
	Interrupt Request	Carries interrupt flow control information (also called Link Service message).
Acknowledgment	Data Acknowledgment	Acknowledges receipt of either a Connect Confirm message or one or more Data Segment messages, and optionally an Other Data message.
	Other Data Acknowledgment	Acknowledges receipt of one or more Interrupt, Data Request or Interrupt Request messages.
	Connect Acknowledgment	Acknowledges receipt of a Connect Initiate message or Retransmitted Connect Initiate message.

(to be continued)

Table 1 (Cont.)
NSP Messages

Type	Message	Description
Control	Connect Initiate	Carries a logical link connect request from a Session Control module.
	Connect Confirm	Carries a logical link connect acceptance from a Session Control module.
	Disconnect Initiate	Carries a logical link connect rejection or disconnect request from a Session Control module.
	No Resources	Sent when a Connect Initiate message (or Retransmitted Connect Initiate message) is received and there are no resources to establish a new port (also called Disconnect Confirm message).
	Disconnect Complete	Acknowledges the receipt of a Disconnect Initiate message (also called Disconnect Confirm message).
	No Link	Sent when a message is received for a non-existing link (also called Disconnect Confirm message).
	No Operation	Does nothing (included for compatibility with NSP V3.1).

2.6 Major NSP Functions

This section summarizes the operation of the major NSP functions which include:

1. Establishing and destroying logical links (Section 2.6.1)

2. Error control (Section 2.6.2)

3. Flow control (Section 2.6.3)

4. Segmentation and reassembly of user data messages (Section 2.6.4)

2.6.1 Establishing And Destroying Logical Links - A source NSP and a destination NSP exchange messages to establish and destroy (in other words, to connect and disconnect) logical links. Figures 3 through 7 summarize the message exchanges. The calls in capital letters under Session Control headings are names of interface functions as described in Section 3.1. The message exchanges below will take place correctly in an implementation, if the algorithms in Section 4 are followed.

```
.---------------------------------.       .----------------------------.
|              Source             |       |   Destination              |
|---------------------------------|       |----------------------------|
| Session        |                |       |                  |Session  |
| Control        | NSP            |       |   NSP            |Control  |
|---------------------------------|       |----------------------------|
|                |                |       |                  |         |
| CONNECT-XMT -->Connect     ------------->                  |         |
|                |   Initiate     |       |                  |         |
|                |                |       |                  |         |
|                |           <------------- Connect          |         |
|                |                |       |  Acknowledgment  |         |
|                |                |       |                  |         |
|                |           <------------- Connect          |         |
|                |                |       |  Confirm<-------- ACCEPT   |
|                | Data           |       |                  |         |
|                |  Acknowledgment ------->                  |         |
|                |                |       |                  |         |
`---------------------------------'       `----------------------------'
```

Figure 3 Connection with Acceptance

```
.--------------------------------.        .---------------------------.
|                Source          |        |  Destination              |
|--------------------------------|        |---------------------------|
| Session      |                 |        |                 |Session  |
| Control      | NSP             |        |  NSP            |Control  |
|--------------------------------|        |---------------------------|
|              |                 |        |                 |         |
| CONNECT-XMT -->Connect      ------------->                |         |
|              |   Initiate      |        |                 |         |
|              |                 |        |                 |         |
|              |              <------------ Connect         |         |
|              |                 |        |  Acknowledgment |         |
|              |                 |        |                 |         |
|              |              <------------ Disconnect      |         |
|              |                 |        |  Initiate<------- REJECT  |
|              | Disconnect      |        |                 |         |
|              |  Complete    ------------>                 |         |
|              |                 |        |                 |         |
`--------------------------------'        `---------------------------'
```

Figure 4 Connection with Rejection

```
.-------------------------------.          .--------------------------.
|                Source         |          |  Destination             |
|-------------------------------|          |--------------------------|
| Session      |                |          |                |Session  |
| Control      | NSP            |          |  NSP           |Control  |
|-------------------------------|          |--------------------------|
|              |                |          |                |         |
| CONNECT-XMT -->Connect     --------------->               |         |
|              |   Initiate     |          |                |         |
|              |                |          |                |         |
|              |             <-------------- No Resources   |         |
|              |                |          |                |         |
`-------------------------------'          `--------------------------'
```

Figure 5 Connection Attempt with No Resources

```
.--------------------------------.          .---------------------------.
|               Source           |          |  Destination              |
|--------------------------------|          |---------------------------|
| Session        |               |          |                 |Session  |
| Control        | NSP           |          |  NSP            |Control  |
|--------------------------------|          |---------------------------|
|                |               |          |                 |         |
| CONNECT-XMT -->Connect      --------.     |                 |         |
|                |  Initiate     |    |  (Connect Initiate    |         |
|                |               |    |   returned to sender  |         |
|                |            <-------'   by Routing)         |         |
|                |               |          |                 |         |
`--------------------------------'          `---------------------------'
```

Figure 6 Connection Attempt with No Communication

```
.--------------------------------.          .---------------------------.
|               Source           |          |  Destination              |
|--------------------------------|          |---------------------------|
| Session        |               |          |                 |Session  |
| Control        | NSP           |          |  NSP            |Control  |
|--------------------------------|          |---------------------------|
|                |               |          |                 |         |
| CONNECT-XMT -->Disconnect   --------------->                |         |
|                |  Initiate     |          |                 |         |
|                |               |          |                 |         |
|                |            <-------------- Disconnect      |         |
|                |               |          |  Complete       |         |
|                |               |          |                 |         |
`--------------------------------'          `---------------------------'
```

Figure 7 Disconnection

2.6.2 Error Control - NSP uses a basic acknowledgment mechanism to ensure that messages are delivered. NSP does this for each of the four data messages listed in Table 1, Section 2.5.

On a logical link, the four data messages can be thought of as moving in two subchannels. One contains Data Segment messages, the other contains Interrupt, Data Request, and Interrupt Request messages (collectively known as Other-Data).

Messages in each subchannel are numbered sequentially by the transmitting NSP. The receiving NSP returns an acknowledgment quickly. Otherwise, the transmitting NSP retransmits the message. It is not necessary to acknowledge each message individually. Acknowledgment of a given numbered message implies acknowledgment of all messages with a lower number (modulo the maximum message number).

Figure 8 depicts the segment acknowledgment operation. Section 8 specifies the format of the acknowledgment messages.

Figure 8 Segment Acknowledgment Operation

```
.-.
|1| The data-transmitting NSP assigns a transmit number to a message,
`-' transmits the message, and starts a timer.
```

```
.-------------------.                          .-------------------.
|                   |  .---------------------. |                   |
|                   |  | transmit number = n | |                   |
|                   |--| Data Segment Message|-->|                 |
| Data-transmitting |  `---------------------' | Data-receiving    |
|        NSP        |                          |       NSP         |
|                   |  .---------------------. |                   |
|                   |  | transmit number = m | |                   |
|                   |--| "Other Data" Message|-->|                 |
|                   |  `---------------------' |                   |
`-------------------'                          `-------------------'
                          Data Subchannels
```

```
.-.
|2| If the timer times out, the message is retransmitted.
`-'
```

```
.-.
|3| If the timer does not time out, and the flow control mechanism
`-' allows another message to be sent, the data-transmitting NSP
    assigns the transmit number plus one to the next data message
    transmitted in that subchannel.
```

```
.--------------------.                            .----------------.
|                    |  .----------------------.  |                |
|                    |  | transmit number = n+1|  |                |
|                    |--| Data Segment Message |-->|               |
| Data-transmitting  |  `----------------------'  | Data-receiving |
|        NSP         |                            |      NSP       |
|                    |  .----------------------.  |                |
|                    |  | transmit number = m+1|  |                |
|                    |--| "Other Data" Message |-->|               |
|                    |  `----------------------'  |                |
`--------------------'                            `----------------'
                          Data Subchannels
```

```
.-.
|4| When the message with the first transmit number is received by
`-' the data-receiving NSP, it returns that number as an acknowledg-
    ment number within the first acknowledgment.
```

|5| If the next data message transmit number received is equal to the current acknowledgment number plus one, the data-receiving NSP accepts the data message, incrementing the acknowledgment number. It then sends the new receive acknowledgment number back to the data-transmitting NSP within an acknowledgment message.

```
.------------.   .- - - - - - - - .   .---------------.  .-----------.
|            |   . receive ack.   .   | receive ack.  |  |           |
|            |   . number = n     .   | number = n + 1|  |           |
|            |<--. Data           .--| Data           |--|           |
|            |   . Acknowledgment .   | Acknowledgment|  |           |
|   Data-    |   . Message*       .   | Message       |  |   Data-   |
|transmitting|   `- - - - - - - - '   `---------------'  | receiving |
|    NSP     |         .------------------.               |    NSP    |
|            |         | receive ack.     |               |           |
|            |         | number = m       |               |           |
|            |<--------| "Other Data"     |---------------|           |
|            |         | Acknowledgment   |               |           |
|            |         | Message          |               |           |
`------------'         `------------------'               `-----------'
```

Data Subchannels

* The data-receiving NSP might not send an acknowledgment for each data message received. The receive acknowledgment number implies that all previous numbers were received.

|6| However, if the data-receiving NSP receives a data message transmit number less than or equal to the current receive acknowledgment number for that subchannel, the data segment is discarded. The data-receiving NSP sends an acknowledgment back to the data-transmitting NSP. The acknowledgment contains the receive acknowledgment number.

|7| If the data-receiving NSP receives a data message transmit number greater than the current receive acknowledgment number plus one for that subchannel, the data segment may be held until the preceding segments are received or it may be discarded.

Figure 8
Segment Acknowledgment Operation

2.6.3 Flow Control - Flow control is the mechanism that determines when to send an NSP Data Segment or Interrupt message. This mechanism, along with the error control mechanism, coordinates the flow of data on a logical link from transmit buffers in one node to receive buffers in another node.

Flow control is performed separately for normal and interrupt data. Flow control operates symmetrically for data flow in each direction on a logical link.

2.6.3.1 Normal Data Flow Control - Flow control requires two algorithms for normal data:

1. An implementation-dependent algorithm executed by a data-receiving NSP that determines both when to send a request message and the count value to be put into a request message.

2. An algorithm executed by a data-transmitting NSP that determines if a Data Segment message may be sent.

In addition, an "on/off" control mechanism may be used by a data-receiving NSP to indicate to a data-transmitting NSP that Data Segment messages may or may not be sent.

On/off control. On/off control is independent of the request count control. It operates as follows: Each Data Request message contains a "send/do not send" indicator. When the error control mechanism in a data-transmitting NSP has received and accepted a Data Request message, the value of the "send/do not send" indicator is saved in the data base associated with the logical link. When the value is "do not send," the data-transmitting NSP may not transmit normal data. When the value is "send," the data-transmitting NSP exercises the flow control mechanisms described next.

Request count flow control. During logical link formation, the NSP at each end of the link determines the kind of flow control it expects when acting as a data receiver. The term "data-receiving NSP" means an NSP acting as a data receiver. There is a choice of:

- No flow control

- Segment flow control

- Session Control message flow control

The choice of flow control is indicated via fields in Connect Initiate, Retransmitted Connect Initiate and Connect Confirm messages.

Each data-transmitting NSP must accept the type of flow control the data-receiving NSP expects. Note, Message flow control is obsolescent and will be eliminated at a future time.

A data-transmitting NSP maintains a "transmit request count" variable for normal data in the data base associated with each logical link. When the error control mechanism receives and accepts a Data Request message, flow control adds the count value from the message to the appropriate transmit request count. The count values contained in the request messages may be zero, positive, or, in some cases, negative. This additive scheme works because the request messages are error-controlled; it would not work otherwise.

No flow control. If the data-receiving NSP selected "no flow control," the data-transmitting NSP may transmit data at any time (subject to the "on/off" constraint).

```
 .-.
 |1| NSP/Node A send a Connect Initiate message to NSP/Node B:
 `-'

     .------------------.
     | Connect Initiate |---------------->
     `------------------'

 .-.
 |2| NSP/Node B, having received the Connect Initiate message, returns
 `-' a Connect Confirm message.  A field in the message indicates that
     NSP/Node B expects segment flow control:
                                           .-------------------------.
                          <----------------| Connect         :  :    |
                                           | Confirm         :  :    |
                                           `-------------------A-----'
                         "I want segment flow control"---------'

 .-.
 |3| NSP/Node A's data base has the initial value of 0 for its request
 `-' count variable for normal data segment flow control:

     .-----------------.
     |  FLOWrem_dat=0  |
     |  .-----------.  |
     `--'           `--'

 .-.
 |4| NSP/Node B sends Data Request message containing a flow control
 `-' value that indicates the number of Data Segment messages NSP/Node
     B can receive.  (NSP/Node B executes an implementation-dependent
     algorithm to determine this value):
                                           .-------------------------.
                           <---------------| Data           :        |
                                           | Request        :        |
                                           `------------------A------'
                          "I can receive n messages"----------'

 .-.
 |5| NSP/Node A sets FLOWrem_dat to n:
 `-'
     .-----------------.
     |  FLOWrem_dat=n  |
     |  .-----------.  |
     `--'           `--'
```

```
.-.
|6| NSP/Node A executes algorithm to determine if Data Segment
`-' Number 1 can be sent (highest acknowledged Data Segment message
    plus the current request count must be greater than or equal to
    the number of the next Data Segement message sent):

                *
              .   .
            .       .
          *  0+n>=1?  *--NO--can't send
            .       .
              .   .
                *
                |
               YES
                |
               SEND
.-.
|7| The answer to the above is YES, so NSP/Node A sends Data Segment
`-' number 1:
    .----------------.
    | Data Segment 1 |----------------->
    `----------------'
.-.
|8| NSP/Node B acknowledges receipt of first data segment:
`-'
                                         .------------------------.
                       <-----------------| Data Acknowledgment    |
                                         | of segment number 1    |
                                         `------------------------'
.-.
|9| NSP/Node A subtracts 1 from its flow control request count:
`-'

    .-----------------.
    | FLOWrem_dat=n-1 |
    |  .-----------.  |
    `--'           `--'
```

Figure 9 Example of Segment Flow Control for Normal Data on a Logical Link (shown in one direction only)

<u>Segment flow control</u>. Figure 9 shows an example of the operation of segment flow control. Segment flow control operates in the following manner: If the data-receiving NSP selected the segment flow control mechanism when the logical link was formed, the highest numbered Data Segment message that may be transmitted is the one whose number is equal to the sum of the highest numbered Data Segment message that has been acknowledged (via the error control mechanism) by the data-receiving NSP plus the current value of the request count. The data-transmitting NSP decrements its request count variable by one for each Data Segment message acknowledged by the data-receiving NSP.

The count values that can be contained in Data Request messages may be negative. This means that the permission to transmit a particular Data Segment message (even if it has been previously transmitted) may be withdrawn by the receiver. This, in turn, causes an interaction between the flow control and error control mechanisms. Specifically, it is not necessary for the error control mechanism to maintain an active retransmission timer for a Data Segment message that has been transmitted at least once but for which permission to transmit (in other words, to retransmit) has been withdrawn.

Session Control message flow control. If the data-receiving NSP selected Session Control message flow control, the data-transmitting NSP cannot send a segment if the number of end-of-message segments between the highest acknowledged segment and the segment in question (exclusive) is greater than or equal to the count. The data-transmitting NSP decrements its request count variable by one for each Data Segment message that is an end-of-message segment acknowledged by the data-receiving NSP.

The mechanism for Session Control message flow control does not interact closely with the error control mechanism (unlike the mechanism for segment flow control). Once a Data Segment has been given permission to be transmitted, the permission will never be withdrawn.

2.6.3.2 Interrupt Data Flow Control - All NSPs use interrupt data flow control for moving interrupt data. This mechanism is similar in operation to the Session Control message flow control mechanism. Interrupt message request counts are carried in Interrupt Request messages. The counts are additive and may not be negative. The interrupt-transmitting NSP can, therefore, maintain an interrupt transmit request count. When a logical link is established, there is an implicit request of one interrupt message. The interrupt transmitting NSP cannot send an Interrupt message if the number of Interrupt messages between the highest acknowledged Other Data message and the Interrupt message in question is greater than or equal to the count. The interrupt-transmitting NSP decrements its request count variable by one for each Interrupt message acknowledged by the interrupt-receiving NSP.

2.6.4 Segmentation And Reassembly Of User Messages - Because of network constraints such as available buffer sizes and transmission error characteristics, user messages in Session Control buffers cannot always be sent in one piece. A data-transmitting NSP breaks up data contained in a single Session Control buffer into segments. A data-receiving NSP reassembles the segments. The data-transmitting

NSP transmits the segments by means of Data Segment messages. The data-receiving NSP puts the segments from the Data Segment messages into the correct sequence. These messages contain user data as well as NSP header information. The segment acknowledgment scheme (Figure 8, Section 2.6.2) ensures that all data segments are received. The data-transmitting NSP must know the maximum length of a Data Segment. This length is the lesser of:

1. The size of a transmit buffer in the source node. This size cannot be larger than the node's Routing Layer will permit.

2. The maximum length that the data-receiving NSP can receive. The SEGSIZE field in the Connect Initiate and Connect Confirm messages, exchanged when the logical link was formed, contains this information.

The data-receiving NSP orders the data segments using the sequence number contained in the Data Segment message and end-of-message information. When Data Segments have been received out of sequence, they are either discarded or stored temporarily in a cache buffer (Section 2.7.1).

This document does not specify the detailed processes of segmentation and reassembly. However, Sections 6.5 and 6.8 provide a model for implementation and Appendixes B and C provide examples.

2.7 Functional Components

In its relation to DNA, NSP can be considered a "black box," which interfaces to Session Control and Routing by defined interfaces and with other NSP modules by the Network Services Protocol. The functional components in this section and in Sections 5 and 6 describe the operation of this "black box" by means of a sample implementation. Any other implementation with equivalent operation is also a legitimate NSP implementation.

NSP consists of data bases, buffer pools, and modules. Brief descriptions of each follow in this section. Section 5 details the data base and buffer pool specifications. Section 6 specifies in detail the operation of the NSP modules with a model implementation written in a high-level, colloquial computer language. Figure 10 shows the interrelationship of the NSP components.

2.7.1 Data Bases And Buffer Pools - The following is a model of the NSP data bases:

NSP internal data base. The NSP internal data base contains NSP's internal variables and parameters. Variables are values defined by NSP. Variables change automatically during the operation of NSP. Parameters are values defined by the Network Management interface. Parameters can be read and sometimes set by the user. Many parameters are static in the sense that they remain set until the user changes them.

Session Control port data base. The Session Control port data base contains the port variables that NSP uses to manage a logical link. When a logical link is created, NSP allocates a Session Control port to it. When the link is destroyed, NSP releases the port back to the port data base as a free port.

Reserved port data base. The reserved port data base contains the port variables reserved for NSP's internal use. NSP uses these to manage the sending of messages that do not map onto the Session Control port data base.

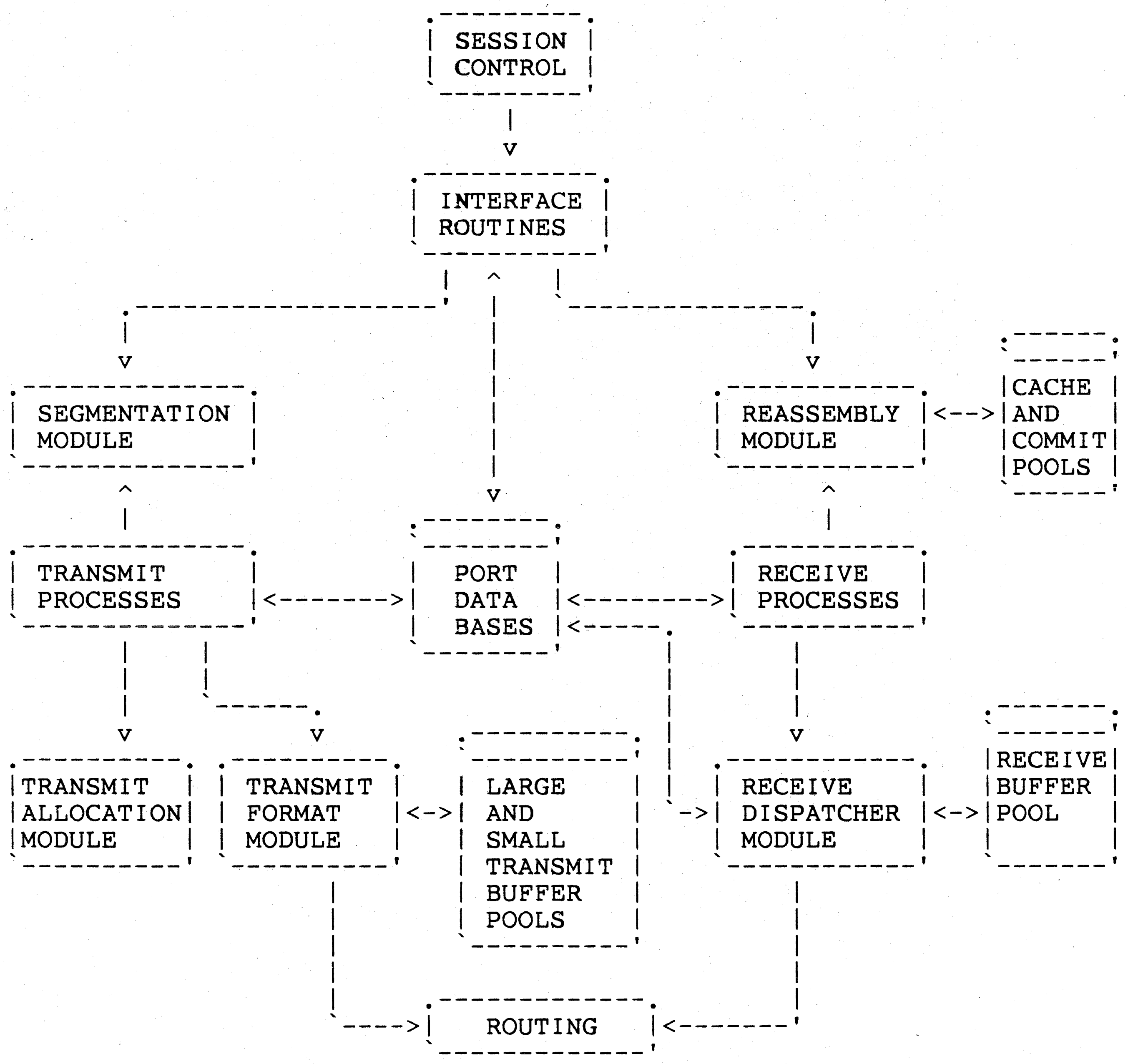


Figure 10 Interrelationship of NSP Components

<u>Node data base</u>. The node data base contains a collection of node descriptors. A node descriptor is required for each remote node to which a logical link is established. A node descriptor contains variables and counters pertaining to communications with that node (for example, the estimated round trip communications delay, traffic usage and error counters).

<u>Large and small transmit buffer pools</u>. The large and small transmit buffer pools contain large and small transmit buffers. Large transmit buffers transmit Connect Initiate (or Retransmitted Connect Initiate)

messages or Data Segment messages. Small transmit buffers transmit all other NSP messages. An implementation may choose to use a single transmit buffer pool for all NSP messages.

Receive buffer pool. The receive buffer pool contains a collection of receive buffers required to receive an NSP message from Routing.

Commit buffer pool. The commit buffer pool is an optional pool which contains commit buffers used for data that the receiving node may commit to storage even in the absence of receive buffers supplied by Session Control. Such data may be acknowledged to its transmitter.

Cache buffer pool. The cache buffer pool is an optional pool which contains a collection of cache buffers. Cache buffers hold received Data Segment messages that cannot be acknowledged either because they are out of order or because there is no storage in either a commit buffer or a Session Control receive buffer for them.

Event buffer pool. The event buffer pool contains buffers that may be queued to NSP's event queue for reading by Network Management.

2.7.2 Modules - NSP modules perform specialized functions. There are two kinds of modules:

1. A process is a module that is independent of other modules, but uses the services of some other modules. It is designed as if it were executing on a processor dedicated to it.

2. A routine is a module that provides functions for one or more processes, but does not have a context of its own.

In general, processes communicate with routines by means of calls and with other processes by means of shared variables, usually a port. The mechanisms that synchronize two processes to prevent common entry to a critical region are not explicitly defined.

The NSP process and routine modules are described below. Note that these are functional descriptions of components. Implementations need not be structured exactly as outlined in these descriptions.

Interface routines. The interface routines handle all Session Control calls (Section 3.1).

Receive dispatcher routine. The receive dispatcher manages the receive buffer pool. It polls Routing for received messages, parses them, maps them onto ports, and returns message contents to the appropriate NSP process.

Receive processes. These receive and handle NSP messages from the End Communications Layer at remote nodes and help manage logical link states. Each Session Control port has a set of these processes.

Reassembly routine. The reassembly routine determines the flow control policy, maintains the cache and commit buffer pools, and reassembles received Data Segment messages into Session Control receive buffers.

Transmit processes. These transmit (and retransmit, if necessary) NSP messages to End Communications Layers at remote nodes. Each Session Control port has a set of these processes.

Transmit format routine. The transmit format routine maintains large and small transmit buffer pools and formats outgoing messages. It queues messages to Routing. It polls Routing to get "transmit complete" notifications.

Segmentation routine. This segments data in Session Control transmit buffers into a form suitable for transmission in Data Segment messages.

Transmit allocation routine. The transmit allocation routine receives requests for permission to transmit from the transmit processes. It allocates permission to transmit in a way that guarantees that when more than one logical link is established to the same destination at the same time each provides useful commuaication services.

3.0 NSP INTERFACES

This section describes the three external NSP interfaces:

1. Session Control interface
2. Network Management interface
3. Routing interface

The interface functions are described as calls to subroutines in the following format:

    FUNCTION (input; output)

Descriptions of input and output then follow unless previously given.

In general, there are two types of subroutines: those performing a function that is completed immediately, and those queueing a buffer for transmitting or receiving data.

For buffer-queueing calls, additional calls are defined to allow polling to obtain "buffer returned" notifications. A "buffer" argument denotes a system-dependent buffer descriptor that contains location and length information. A "port id" is a system-dependent number identifying a port. Although not described in the following functions, an invalid port identifier causes an error.

Note that an implementation is not required to code the interfaces as calls to subroutines. The calls specify functions only.

It may be useful to refer to the port state descriptions in Section 4.1 when studying the following interface functions.

3.1 Session Control Interface

This interface allows NSP to provide Session Control with the logical link service. This service allows Session Control to create one or more logical links to one or more other Session Control modules in the same network.

In the interface descriptions, the terms "source" and "destination" distinguish the requester of a function from the receiver of the request. The source and destination can be within a single Session Control module or in two separate Session Control modules. Thus, at a single node, a Session Control module can communicate with itself via a logical link; between two nodes, two Session Control modules can communicate with each other via a logical link.

The calls, described by function, are as follows:

STATUS (; NSP status)

returns: NSP is halted.

NSP is running; minimum receive buffer size (NSPbuf -- Table 3, Section 5.1) returned

This function reads the status of NSP and obtains a minimum receive buffer size if NSP is running. This is the one Session Control interface function that does not involve the use of a logical link.

OPEN (source, buffer; return)

source: a 16-bit buffer to contain the logical link requester node address when this node receives a connect request

returns: port allocated and port identifier returned

port not allocated -- insufficient resources

port not allocated -- NSP halted

This function allocates a port in NSP for receiving a logical link connect request. The source variable receives the node address of the requesting node; the buffer receives the incoming connect data. When the port state indicates an incoming connect request is received, NSP places the source node address in the source variable and the incoming data in the buffer.

<u>CLOSE</u> (port id)

This function deallocates a port. When a port is closed, NSP immediately returns all transmit and receive buffers to Session Control (see DATA-XMT and DATA-RCV calls). Once a port is closed, its associated port identifier is undefined. Any subsequent call issued with such a port identifier results in an error return.

Session Control may close a port at any time regardless of the port's state. However, doing so may create ambiguities for the Session Control module at the other end of the logical link.

<u>CONFIDENCE</u> (port id; confidence)

returns: network probably connected

network probably disconnected

port not in RUNNING, CONNECT-CONFIRM, DISCONNECT-REJECT, or DISCONNECT-INITIATE state

This function obtains NSP's assessment of connectivity. NSP periodically tests a logical link once it is formed to ascertain if the physical path supporting the link is connected. The result of this testing is the probable state of connectivity. It is not a guaranteed state.

Session Control may issue this call to determine when to disconnect a link on behalf of a program at the user level.

<u>STATE</u> (port id; state)

returns: the state of the associated logical link

This function returns the state of a port that is not CLOSED.

Because NSP's operation is not necessarily synchronized with that of Session Control, it is possible that this call will not detect every state transition. This is especially the case for state transitions that occur very quickly. However, this is not a problem because the intervening undetected states can be logically deduced.

CONNECT-XMT (destination [,circuit [,nexthop]], buffer; return)

destination: destination node address

circuit: an internal NSP mechanism selector used to enable loop testing. Circuit is either unspecified (for normal use) or a system-dependent circuit number representing the circuit NSP is to use for its messages establishing this logical link (for Network Management loop tests).

nexthop: a node number (required if circuit is a broadcast circuit)

buffer:: contains connect data

returns: port allocated; port id returned

port not allocated -- insufficient resources

port not allocated -- NSP halted

This function allocates a port and requests a logical link connection. After a logical link has been successfully formed, Session Control can put a load on a particular physical link for loop test purposes provided that the circuit argument specified the physical link. This enables testing of the physical link and all of the DECnet modules from Session Control or higher layers by sending and receiving data on the resulting logical link. For normal use, the circuit argument is set to "unspecified."

CONNECT-STATUS (port id, buffer; return)

returns: connect request accepted by destination -- port in RUNNING state; accept data returned in buffer

connect request rejected by destination -- port in REJECTED state; reject data returned in buffer

port in neither RUNNING nor REJECTED state

This function obtains accept or reject data returned as a result of a previous connect request. If the return is one of the first two, NSP returns any available accept or reject data. Once this is done, an NSP implementation may discard its copy of the accept or reject data so that a subsequent connect status function would not return data.

In cases where state transitions occur very rapidly, Session Control may not be able to perceive some intervening states. Consequently, this call may not be accepted (see Section 4.1).

Accept data will be lost if the rapid state transitions end with a transition to the DISCONNECT-NOTIFICATION state and this call was never executed in the RUNNING state. No data is lost otherwise.

If the connect request is accepted, up to 16 bytes of accept data may be returned in the buffer. If the connect request was rejected, up to 18 bytes of reject data may be returned in the buffer (see the ACCEPT and REJECT calls).

<u>ACCEPT</u> (port id, buffer; return)

returns: link accepted

port not in CONNECT-RECEIVED state

This function accepts a connect request from a remote Session Control module. The call supplies a buffer containing up to 16 bytes of accept data.

<u>REJECT</u> (port id, value, buffer; return)

returns: link rejected

port not in CONNECT-RECEIVED state

This function rejects a connect request from a Session Control module. The call supplies a 2-byte value and a buffer containing up to 16 bytes of reject data.

<u>DISCONNECT-XMT</u> (port id, value, buffer; return)

returns: call accepted

call rejected -- port not in RUNNING state

This function requests the disconnection of a logical link that is in the RUNNING state. The call supplies a 2-byte value and a buffer containing up to 16 bytes of disconnect data.

The remote Session Control module receives any data transmitted by the disconnecting Session Control module prior to this call. Session Control disconnects a link when it has no more data to send and wants to ensure that the link will be properly disconnected, not aborted.

ABORT-XMT (port id, value, buffer; return)

returns: call accepted

call rejected -- port not in RUNNING state

This function requests the immediate disconnection of a logical link that is in the RUNNING state. The remote Session Control module may not receive all previously transmitted data before receiving the abort notification.

The call supplies a 2-byte value and a buffer containing up to 16 bytes of abort data.

DISCONNECT-RCV (port id, value, buffer; return)

returns: disconnect data available

no disconnect data available

port not in DISCONNECT-NOTIFICATION state

This function receives disconnect data returned to the local Session Control module as a result of a DISCONNECT-XMT or ABORT-XMT call from the remote Session Control module. Session Control detects a logical link disconnection or an abort when a STATE call returns a DISCONNECT-NOTIFICATION. A 2-byte value and up to 16 bytes of data may be returned in the buffer.

DATA-XMT (port id, buffer, xmtflag; return)

xmtflag: a flag indicating whether the last byte in the buffer is the last byte of a Session Control message. Its value is one of:

- end-of-message
- not-end-of-message

Section 2.4.4 describes data flow.

returns: buffer queued

buffer not queued -- insufficient resources

port not in RUNNING state

This function queues a transmit buffer to a port for transmitting normal data on a logical link. NSP refuses to queue the buffer either if it lacks the resources to do so or if the port is not in the RUNNING state.

XMT-POLL (port id; return)

returns: no tranmsit complete

transmit complete -- buffer descriptor returned

This function returns a transmit buffer to Session Control.

DATA-RCV (port id, buffer, rcvflag; return)

rcvflag: a flag indicating whether data truncation is allowed. It may have either of the following values:

- o no truncation allowed
- o truncation allowed

returns: buffer queued

buffer not queued -- insufficient resources

buffer not queued -- buffer too small and no truncation was specified in rcvflag

port not in RUNNING or DISCONNECT-INITIATE state

This function queues a receive buffer to a port to receive normal data. A "buffer too small" return indicates the buffer size is smaller than the minimum receive buffer, NSPbuf (see STATUS).

Session Control may provide a buffer to a port in the DISCONNECT-INITIATE state to avoid a Session Control deadlock in which each end of the logical link is in the DISCONNECT-INITIATE state. However, this is an implementation-dependent issue.

RCV-POLL (port id; return)

returns: no buffer returned (Either no receive buffers are queued to the port or there is no receive data available.)

buffer returned -- no data lost, end-of-message

buffer returned -- data lost, end-of-message

buffer returned -- no data lost, not end-of-message

buffer returned -- data lost, not end-of-message buffer returned empty -- port not in RUNNING, DISCONNECT-INITIATE, DISCONNECT-COMPLETE, or DISCONNECT-NOTIFICATION states.

This function obtains a "receive complete" notification for a receive buffer previously queued via a DATA-RCV call. NSP returns receive buffers along with buffer descriptors to Session Control in the order in which data was placed in them.

A data-transmitting NSP segments data given to it by the DATA-XMT call and sends each segment separately through the network. A segment containing data given to NSP with an "end-of-message" flag is so marked. A data-receiving NSP receives these segments and places the data in Session Control receive buffers given to NSP by the DATA-RCV function. The sequence of packets flowing from a data-transmitting NSP to a data-receiving NSP constitutes the network form described in Section 2.4.4.

If a data-receiving Session Control module gives NSP each receive buffer with the rcvflag set to "no truncation allowed" on the DATA-RCV call, then NSP attempts to place the data, in order, from one or more segments of a single Session Control message into each receive buffer. A receive buffer is always returned with a "no data lost" indication and is returned with an "end-of-message" indication if and only if the last segment of data placed in it was marked as an "end-of-message" segment. Note that two DATA-XMT calls by the data-transmitting Session Control module, one with data that is not marked "end-of-message" and the second with no data but marked "end-of-message," may result in data and status being given to the data-receiving Session Control either in two buffers, as given to NSP by the data-transmitting Session Control module, or in a single buffer containing the data and marked as "end-of-message."

If a data-receiving Session Control module gives NSP each receive buffer with the rcvflag set to "truncation allowed," then NSP either fills the receive buffer or puts data into it up to and including the data in a segment marked as "end-of-message" whichever comes first. If a receive buffer is filled first, then NSP continues to receive and discard data segments up to and including the first one marked as "end-of-message." In either case, the receive buffer is returned as an "end-of-message" on the RCV-POLL call. In the case where data was discarded, the receive buffer is returned with a "data lost" indication. The only time a buffer given with a "truncation allowed" rcvflag is returned as "not-end-of-message" is when the logical link is disconnected by the data-transmitting Session Control module with a partially transmitted message.

A data-receiving Session control module may mix calls with the "truncation allowed" rcvflag with calls with the "no truncation allowed" rcvflag.

If Session Control closes a port that has receive buffers queued via the DATA-RCV call, NSP returns these buffers immediately.

INTERRUPT-XMT (port id, buffer; return)

returns: data accepted

data not accepted

port not in RUNNING state

This function sends up to 16 bytes of high priority data to the destination Session Control module. The data has no sequential relationship to normal data transferred on a logical link. NSP may refuse a request to send interrupt data if it is unable to queue the data internally. The buffer may be up to 16 bytes long.

INTERRUPT-RCV (port id, buffer; return)

returns: data returned

no data returned

port not in RUNNING state

This function obtains available interrupt data. Interrupt data is delivered in the order transmitted by the INTERRUPT-XMT function. Interrupt data has no sequential relationship to normal data transferred on a logical link.

3.2 Network Management Interface

Network Management can perform the following functions using NSP's Network Management interface:

1. Initialize or halt NSP.

2. Read and set some of NSP's local parameters and variables.

3. Read NSP's remote node variables and counters. Clear NSP's remote node counters.

4. Read (one event at a time) and clear NSP's event queue.

The interface is modeled as a collection of functions provided by subroutines. Each call represents a specific function. In each return, the variable in parentheses is its name appearing in the data base descriptions in Tables 3 and 4, Section 5.

Many of the calls pertain to either local or remote NSPs. Actually, the "remote" NSP is the local NSP if the logical link is made within a single node.

All the variables read, set, or cleared by the following Network Management functions are locally kept variables. Thus, a set of NSP variables for each remote node to which there is an active logical link is kept at the local node. NSP does not guarantee that any information will be available when there are no logical links to the specified remote node.

Implementations of Network Management are not required to return every parameter, variable, or counter listed here.

The Network Management interface functions are as follows:

INITIALIZE

This function initializes the local NSP.

HALT

This function halts NSP. The call has the following effects:

1. NSP closes all ports.

2. NSP will refuse to open a new port.

3. NSP returns all Session Control buffers to Session Control.

4. NSP freezes its counters and event queue.

LOCAL-READ-ADDRESS (; address)

return: NSP's node address (NSPself)

This function reads NSP's node address.

LOCAL-READ-INACTIVITY (; inactivity)

return: NSP's inactivity timer (NSPinact_tim)

This function returns the interval after which, if there is no data going in either direction on a link, NSP checks the link. NSP checks the link by sending a Data Request message (Table 1) to the remote NSP. This message does not change flow control parameters, but does require an acknowledgment.

LOCAL-READ-DELAY (; delay)

return: NSP's delay factor (NSPdelay)

This function returns the number by which to multiply one sixteenth of the estimated round trip delay to a node to set the retransmission timer to that node. The round trip delay is used in an NSP algorithm that determines when to retransmit a message (Section 7.3).

LOCAL-READ-WEIGHT (; weight)

return: NSP's delay weight (NSPweight)

This function returns the weight NSP applies to the current round trip delay estimate to a remote node when updating the estimated round trip delay to a node (Section 7.3).

LOCAL-READ-RETRANSMIT (; retransmit)

return: NSP's retransmit threshold (NSPretrans)

This function returns the maximum number of times the source NSP will restart an expired retransmission timer before deciding that the remote node is probably unreachable (see CONFIDENCE in Section 3.1). When this number is exceeded, NSP gives a "probable disconnection" return to a Session Control CONFIDENCE call. Session Control may then disconnect the link on behalf of the end user. The retransmit threshold is called the NODE RETRANSMIT FACTOR in the DNA Network Management Functional Specification.

LOCAL-READ-MAXIMUM-ACTIVE (; maximum ports)

return: NSP's maximum ports number (NSPmax)

This function returns the maximum number of ports NSP has concurrently had assigned to a logical link (in other words, concurrently in a state other than OPEN or CLOSED). This is called NODE MAXIMUM LOGICAL LINKS ACTIVE in the DNA Network Management Functional Specification.

LOCAL-READ-TOTAL (; total ports)

return: NSP's total ports number (NSPtotal)

This function returns the maximum active logical link count for the node. This is the total number of ports that NSP can have in use concurrently. This is called NODE MAXIMUM LINKS in the DNA Network Management Functional Specification.

LOCAL-READ-VERSION (; version number)

return: NSP's version number (NSPversion)

This function returns the local NSP's version number. For this version, the value returned is 4.0.

LOCAL-SET (qualifier, value)

qualifier: one of the following, defined above:

LOCAL-SET-ADDRESS
LOCAL-SET-DELAY
LOCAL-SET-INACTIVITY
LOCAL-SET-MAXIMUM
LOCAL-SET-RETRANSMIT
LOCAL-SET-WEIGHT

value: the new numerical value for the selected parameter or variable.

This function sets NSP local parameters.

REMOTE-READ-DELAY (node; delay)

node: a node address

return: the estimated round trip delay to the remote node (NODEdelay)

This function returns the estimated round trip delay to the remote node.

REMOTE-READ-ACTIVE (node; active)

return: the number of active logical links to the remote NSP (NODEactive)

This function returns the number of ports in a state other than OPEN that NSP has associated with logical links to the remote node. This variable is called NODE ACTIVE LINKS in the DNA Network Management Functional Specification.

REMOTE-READ-BYTES RECEIVED (node; bytes received)

return: the number of user data bytes received (NODEbyt_rcv)

This function returns the total number of user data bytes received from the remote node, including normal, interrupt, connect, accept, reject and disconnect data.

REMOTE-READ-BYTES SENT (node; bytes sent)

return: the number of user data bytes sent (NODEbyt_xmt)

This function returns the total number of user data bytes sent to the remote node.

REMOTE-READ-MESSAGES RECEIVED (node; messages received)

return: the total number of messages received from the remote node (NODEmsg_rcv)

This function returns the total number of all types of NSP messages received from the remote node regardless of their disposition. This includes detected duplicates.

REMOTE-READ-MESSAGES SENT (node; messages sent)

return: the total number of NSP messages sent to the remote node (NODEmsg_xmt)

This function returns the total number of NSP messages sent to the remote node. This includes retransmissions.

<u>REMOTE-READ-CONNECTS</u> <u>RECEIVED</u> (node; connects received)

return: the total number of NSP Connect Initiate messages received from the remote node (NODEcon_rcv)

This function returns the total number of NSP Connect Initiate messages the local node has received from the remote node regardless of their disposition.

<u>REMOTE-READ-CONNECTS</u> <u>SENT</u> (node; connects sent)

return: the number of NSP Connect Initiate messages sent to the remote node (NODEcon_xmt)

This function returns the number of NSP Connect Initiate messages sent to the remote node.

<u>REMOTE-READ-CONNECTS</u> <u>REJECTED</u> (node; connects rejected)

return: the number of received NSP Connect Initiate messages for which there was no open port (NODEcon_rej)

This function returns the number of NSP Connect Initiate messages rejected by the local NSP. This variable is called "received connect resource errors" in the DNA Network Management Specification.

<u>REMOTE-READ-TIMEOUTS</u> (node; timeouts)

return: the number of timeouts that have occurred in waiting for acknowledgments (of any kind) from the remote node (NODEtimeout)

This function returns the number of response timeouts received from the destination NSP.

<u>REMOTE-READ-AND-CLEAR</u> (node, value; qualifier)

value: the numerical value for the corresponding qualifier.

qualifier: one of the following, defined above:

- o REMOTE-READ-BYTES RECEIVED
- o REMOTE-READ-BYTES SENT
- o REMOTE-READ-MESSAGES RECEIVED
- o REMOTE-READ-MESSAGES SENT

- o REMOTE-READ-CONNECTS RECEIVED
- o REMOTE-READ-CONNECTS SENT
- o REMOTE-READ-CONNECTS REJECTED
- o REMOTE-READ-TIMEOUTS

returns: the selected information

This function reads and then clears a node-dependent NSP counter.

<u>READ-QUEUE</u> (; return)

returns: event queue empty

the first entry on the event queue. One of the following events:

invalid message — A received message was discarded because reserved bits or values in the message were used. The beginning of the message is the event data.

invalid flow control — A received Link Service message was discarded because it contained a request count that, when used to compute an accumulated request count, would produce a result out of limits. The message and current request counts are the event data.

data base reused — A node descriptor (Section 5.4) was reused for a different remote node. The contents of the data base are the event data.

events lost — The queue was full when NSP attempted to place an entry in it. One or more events were lost.

This function reads the first entry on an event queue. NSP maintains an internal event queue. The length of the queue is implementation-dependent, but is always at least one. When events (described above in the returns) occur, NSP places entries in the queue. If the queue is full when NSP attempts to place an entry in it, the last entry in the queue changes to "events lost."

The DNA Network Management Functional Specification describes return formats.

CLEAR-QUEUE

This function clears NSP's internal event queue.

3.3 Routing Interface

NSP requires a Routing service for its operation. A Routing service provides NSP with the ability to send datagrams (containing NSP messages) to, and receive datagrams from, any other NSP module in the same DECnet network.

The interface described below appears in the form of calls from an NSP module to a Routing module in the same node.

ROUTING-TRANSMIT (source, destination, returnflag [,circuit[,nexthop]], buffer, tryhard; return)

source: a source node address

destination: a destination node address

returnflag: a Boolean flag to indicate whether or not the packet is to be returned by the Routing service if the destination node is inaccessible. The flag may have one of the following values:

- o false -- do not return packet
- o true -- return packet

circuit: an internal NSP mechanism selector (used for loop testing). One of:

- o unspecified
- o a circuit that is the circuit on which Routing should direct this datagram. If circuit specifies a broadcast circuit, then nexthop must also be specified.

nexthop: a node number

buffer: a buffer containing a packet

tryhard: a Boolean flag to indicate whether or not the packet is to be sent to the destination node using the

Routing Layer's tryhard algorithm. When this flag is true, NSP is instructing the Routing Layer to make a maximal effort to deliver the packet. When false, NSP is instructing the Routing Layer to deliver the packet using a less costly and potentially less reliable mechanism. The normal default is false.

returns: buffer queued

buffer not queued

This function queues a transmit buffer to Routing. Routing rejects this call (in other words, does not queue a packet) only if it has insufficient resources to queue the buffer. If the destination node is currently unreachable, then Routing accepts the buffer, although it may return the buffer immediately (see ROUTING-CHECK-TRANSMIT-BUFFER call, following). The "circuit" argument has the same meaning as that in the CONNECT-XMT call (Section 3.1).

ROUTING-CHECK-TRANSMIT-BUFFER (buffer, return;)

returns: all buffers remain queued by Routing

buffer returned to NSP

buffer: a buffer previously given to Routing via a ROUTING-TRANSMIT call

This function checks to see if a previously queued transmit buffer can be returned to NSP.

ROUTING-SUPPLY-RECEIVE-BUFFER (buffer; return)

returns: buffer queued for receive by Routing

buffer not queued for receive by Routing

This function queues a receive buffer to Routing.

ROUTING-CHECK-RECEIVE-BUFFER (source, destination,
[,circuit[,nexthop]], buffer, tryhard; return)

source: a source node address

destination: a destination node address

circuit: an internal NSP mechanism selector (used for loop testing). One of:

- o unspecified
- o a circuit that is the circuit on which Routing should direct this datagram. If circuit specifies a broadcast circuit, then nexthop must also be specified.

nexthop: a node number

returns: all buffers remain queued by Routing

buffer returned with source and destination node addresses -- contains a normal packet

buffer returned -- contains a "return to sender" packet

This function checks to see if a previously queued receive buffer can be returned to NSP.

4.0 NSP STATES

This section describes the states and state transitions required for normal NSP operation.

4.1 Port States

Whenever Session Control allocates a port, NSP associates the port with a state. This state is represented by a variable in the port data base. The CLOSED state really means that the port no longer exists. This state is necessary in the architecture because the remote port may be placed in the CLOSED-NOTIFICATION state. In some implementations the CLOSED state may be indicated by the absence of an entry. A port has only one state at a time. Table 2 summarizes the normal port states.

Table 2
Port States

(Symbol) State	Explanation
(O) OPEN	The local Session Control has issued an OPEN call which created the port.
(CR) CONNECT-RECEIVED	NSP has received a Connect Initiate message. (The remote port is or was in the CONNECT-INITIATE state.)
(DR) DISCONNECT-REJECT	The local Session Control has issued a REJECT call while the port was in the CONNECT-RECEIVED state.
(DRC) DISCONNECT-REJECT-COMPLETE	NSP has received a Disconnect Complete message while in the DISCONNECT-REJECT state. (The remote port is or has been in the REJECTED state.)
(CC) CONNECT-CONFIRM	The local Session Control has issued an ACCEPT call, while the port was in the CONNECT-RECEIVED state.
(CI) CONNECT-INITIATE	The local Session Control has issued a CONNECT-XMT call, which created this port.
(NR) NO-RESOURCES	NSP has received a No Resources message while in the CONNECT-INITIATE state. (The remote NSP did not have an available port in the OPEN state.)
(NC) NO-COMMUNICATION	NSP has received its own Connect Initiate message while in the CONNECT-INITIATE state because Routing was unable to deliver the message.
(CD) CONNECT-DELIVERED	NSP has received a Connect Acknowledgment message while in the CONNECT-INITIATE state. (The destination port is or has been in the CONNECT-RECEIVED state.)

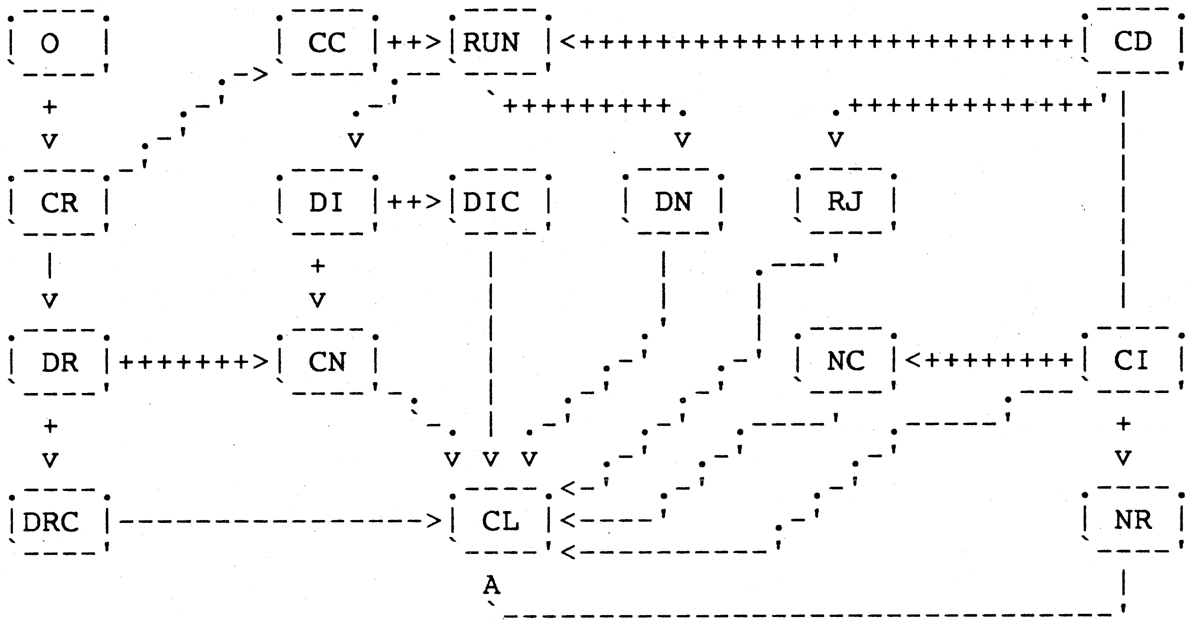
Table 2 (Cont.)
Port States

(Symbol) State	Explanation
(RJ) REJECTED	NSP has received a Disconnect Initiate message while in the CONNECT-INITIATE or CONNECT-DELIVERED state. (The remote port is or has been in the DISCONNECT-REJECT state.)
(RUN) RUNNING	NSP has either received a Connect Confirm message while in the CONNECT-INITIATE or CONNECT-DELIVERED state or received a Data, Data Request, Interrupt Request, Data Acknowledgment or Other Data Acknowledgment message while in the CONNECT-CONFIRM state. The logical link may be used for sending and receiving data. (Either the remote port is or was in the CONNECT-CONFIRM state or the remote port entered the RUNNING state from the CONNECT-DELIVERED state.)
(DI) DISCONNECT-INITIATE	The local Session Control has issued a DISCONNECT-XMT call or an ABORT-XMT call while in the RUNNING state.
(DIC) DISCONNECT-COMPLETE	NSP has received either a Disconnect Complete message or a Disconnect Initiate message while in the DISCONNECT-INITIATE state. (The remote port is or has been in either the DISCONNECT-NOTIFICATION state or the DISCONNECT-INITIATE state.)
(DN) DISCONNECT-NOTIFICATION	NSP has received a Disconnect Initiate message while in the RUNNING state. (The remote port is or has been in the DISCONNECT-INITIATE state.)

Table 2 (Cont.)
Port States

(Symbol) State	Explanation
(CL) CLOSED	The local Session Control has issued a CLOSE call (normally while the local port was in the DRC, DN, DIC, NC, NR or CI state). This is not really a state of the port, but is used for descriptive purposes to indicate that the port is not there.
(CN) CLOSED-NOTIFICATION	NSP has received a No Link message while in the DISCONNECT-INITIATE, DISCONNECT-REJECT, RUN or CONNECT-CONFIRM state. (The remote NSP closed the remote port.)

Figure 11, following, summarizes the normal port state transitions. These transitions correspond with those in Table 2.



LEGEND:

```
.--.
|  |   contains port state (Table 2)
`--'

++++> result from an action by NSP

----> result from a Session Control call
```

NOTES

1. A state from which an exit can be made by a ++++> arrow is a potentially unstable state.

2. A state from which the only exits are ----> arrows are stable states.

3. A state from which an exit can be made only by more than one ++++> arrow is a state from which the exit is non-deterministic.

Figure 11 Port State Diagram

A transition to CLOSE from any state other than those connected by arrows to CL on Figure 11 is equivalent to terminating the logical link while it was in a useful state. Such a transition would occur, for example, when a user process terminates abnormally. This is the only kind of transition that can occur that is not shown in Figure 11.

4.2 Logical Link States

When one Session Control module attempts to form a connection to a second Session Control module NSP places the requesting port in the CONNECT-INITIATE (CI) state. NSP then attempts to associate the local source port with a destination port that is in the OPEN (O) state. If the association between the two ports is successful, the combination of the two port states is the logical link state. The initial logical link state is represented as CI/O.

A logical link can be in only one state at a time. NSP implementations that follow the model in Section 6 will make the transitions correctly.

An NSP may fail to associate two ports. This can happen in the following situations:

- o If the network is disconnected so that the destination system is not accessible. In this case, NSP places the requesting port in the NO-COMMUNICATION state.
- o If there are no ports managed by the remote NSP in the OPEN state. In this case, NSP places the requesting port in the NO-RESOURCES state.
- o If the network is not operating properly (e.g. is badly congested) and succesive retransmissions fail, Sessions Control can detect this by checking the Confidence variable.

Figure 12 shows the normal logical link state transitions.

The only state transitions that are not illustrated in Figure 12 are those that represent the transition of one of the ports to the CLOSED state from a non-terminal state. Such a transition is generally succeeded by a transition of the other port to the CLOSED-NOTIFICATION state. These transitions introduce ambiguity into the diagram of logical link transitions. Figure 12 shows an example of this kind of ambiguity in the exits from the CL/DI and DI/CL states. Figure 12 does not show this complete set of transitions, because there are too many to represent coherently in a single diagram. Moreover, they obscure the transitions that occur during normal operation of a logical link.

Note: In certain unlikely cases a port in the open state may match up with a duplicate request for connection (for example, when the original connection was rejected). The connect data will appear to be valid. If the receiving session control user accepts this false connection the local port will never enter the RUNNING state. However, if communication exists with the remote node, the local port will eventually enter the "CN" state. Because of these cases, a user program which does not wish to process the data from a duplicate connect should accept incoming connections and process the connect data only if the connection enters the RUNNING state.

```
                  .----.    * An exit is made to the CL/CL state from
                  | CI |      this state if the port that is still
                  | O  |      open is closed by Session Control.
                  `----'
                    +
                    v
         .----.   .----.   .----.   .----.     .----.*
         | CI |<--| CI |-->| CI |   | CL |++++>| CL |
         | CC |   | CR |   | DR | ,>| DR |     | CN |
         `----'   `----'   `----' | `----'     `----'
           +        +        +    |    +
           v        v        v    |    v
         .----.   .----.   .----. | .----.*
         | CD |<--| CD |-->| CD | | | CL |
         | CC |   | CR |   | DR | | | DRC|
         `----'   `----'   `----' | `----'
           +                 +    |    ^
           v                 v    |    |
         .----.            .----. | .----.     .----.*
         |RUN |            | RJ |-  | RJ |---->| RJ |
         |CC  |            | DR |++>|DRC |     | CL |
         `----'            `----'   `----'     `----'
           +
           v
         .----.   .----.            .----.     .----.   .----.*
         |RUN |-->|RUN |+++++++++++>| DN |++++>|DN  |-->| DN |
         |RUN |   |DI  |            | DI |     |DIC |   | CL |
         `----'   `----'            `----'     `----'   `----'
           |        |                 |           |
           v        v                 v           |
         .----.   .----.   .----.   .----.        |
         |DI  |-->| DI |++>|DIC |-->| CL |++++    |
         |RUN |   | DI |   |DI  |   | DI |   +    |
         `----'   `----'   `----'   `----'   +    |
           +        +        +        +      +    |
           +        v        v        v      +    |
           +      .----.   .----.   .----.*  +    |
           +      |DI  |++>|DIC |   | CL |   +    |
           +      |DIC |   |DIC |   | CN |   +    |
           +      `----'   `----'\  `----'   +    |
           +        |       \ \              +    |
           v        v        \ \             +    v
         .----.   .----.   .----. \ \        `>.----.*
         | DI |-->| DI |++>| CN |  \ `-------->|CL  |
         | DN |   | CL |   | CL |   \          |DIC |
         `----'   `----'   `----'    `---      `----'
           +        +                    |
           v        +                    v
.----.*  .----.     +++++++++++++++++>.----.*
| CL |<--|DIC |----------------------->|DIC |
| DN |   |DN  |                       |CL  |
`----'   `----'                       `----'
```

Figure 12 Logical Link State Diagram

LEGEND:

```
.--.
|  | contains logical link state consisting of the port states
`--' at both ends of the link (Table 2).

+++> transition caused by NSP

---> transition caused by a Session Control call
```

NOTES

1. A state from which an exit can be made by a +++++> arrow is a potentially unstable state.

2. A state from which the only exits are -----> arrows are stable states.

3. A state from which an exit can be made by more than one +++++> arrow is a state from which the exit is non-deterministic.

4. The logical link states presented above describe the disconnection or abortion of the link from the RUN state when requested by either Session Control module. This is true because the Session Control requesting a disconnection could be either the Session Control that requested the logical link or the module that accepted the logical link.

5. If a logical link enters the DI/RUN or RUN/DI state because of a disconnect request by one of the Session Control modules, then an NSP exit from the DI/RUN or RUN/DI states is possible only if the Session Control module in the RUN state has provided a sufficient number of receive buffers to receive all data transmitted by the other Session Control module. The +++++> arrow exit from either of these states means that NSP guarantees to make exit eventually only if this constraint has been met. Similarly, an NSP exit from the DI/DI state is possible only if one of the Session Control modules sharing the logical link has met this constraint. This constraint does not apply when the DI port state is entered because of an abort request.

Figure 12 Logical Link State Diagram (continued)

5.0 NSP DATA BASES AND BUFFER POOLS

This section specifies the variables and parameters in the data bases and buffer pools described in Section 2.2:

- o NSP's internal data base (Section 5.1)
- o Session Control port data base (Section 5.2)
- o Reserved port data base (Section 5.3)
- o Node data base (Section 5.4)
- o Buffer pools (Section 5.5)

5.1 NSP's Internal Data Base

This data base contains NSP's internal variables and parameters. Network Management can modify some of these (Section 3.2). Table 3 describes the data base.

Table 3
NSP's Internal Data Base

Name	Initial Value	Bit Width	Definition
NSPstate	"halted"	1	State: "halted" or "running"
NSPself	0	16	The node address of this NSP
NSPinact_tim	0+	*	NSP's inactivity time value (in system-dependent units). 0 means "no time value" (Section 7.4).
NSPdelay	2+	8	NSP's "delay factor" (Sect. 6.6)
NSPweight	3+	8	NSP's "round trip delay estimation factor" (Section 6.6).
NSPretrans	5+	8	NSP's "retransmit threshold" for determining confidence in network connectivity for a logical link (Section 7.5).
NSPmax	0	*	The maximum number of ports that NSP has had simultaneously in a state other than OPEN.
NSPversion	4.0	*	NSP's version number.
NSPtotal	*	*	The total number of ports NSP can handle simultaneously.
NSPbuf	*	*	The minimum Session Control rcv. buffer size and the maximum NSP transmit segment size for this implementation (this value is equal to the maximum Routing Layer block size [returned from Routing Layer Read Blocksize.] minus 13, the maximum NSPheader size. See Section 5.5)
NSPtryhard	ceiling of NSPretrans/2	8	NSP's "tryhard threshold" for determining when the tryhard flag should get set (Sect. 5.2)

* This value is not essential to the model implementation.

+ This is a suggested initial value; implementations are not bound to use this as an initial value.

5.2 Session Control Port Data Base

This data base consists of a collection of ports. NSP allocates a port on a Session Control OPEN or CONNECT-XMT call. A port contains the minimal information required to maintain a logical link. Table 4 specifies a Session Control port.

Table 4
Session Control Port

Name	Initial Value	Bit Width	Definition
STATE	O or CI	5	The state of the port.
NODE	0	16	Remote node address.
ADDRloc	0	16	Local link address.
ADDRrem	0	16	Remote link address.
CIRCUIT	0	*	The circuit number to use to transmit messages to Routing. One of: 0 (= "unspecified") a circuit number
NEXTHOP	0	16	Node Number
VERSION	0	*	The version of the remote NSP.
TIMERdat	0	*	Message timer value for Data Segment messages.
TIMERoth	0	*	Message timer value for Interrupt or Link Service messages (data type messages other than Data Segment).
TIMERcon	0	*	Message timer value for Connect and Disconnect messages.
TIMERinact	0	*	Inactivity timer value (Sect. 7.4)
NUMdat	1	12	Number of next Data Segment message to transmit.

Table 4 (Cont.)
Session Control Port

Name	Initial Value	Bit Width	Definition
NUMoth	1	12	Number of next Interrupt or Link Service message to transmit (data type messages other than Data Segment).
NUMhigh	0	12	Number of highest numbered Data Segment message available from Session Control.
NUMsent	0	12	Number of highest numbered Data Segment message sent by local NSP.
ACKxmt_dat	0	12	Number of last Data Segment message acknowledgment sent by local NSP.
ACKxmt_oth	0	12	Number of last Interrupt or Link Service message acknowledgment sent by the local NSP.
ACKrcv_dat	0	12	Number of highest Data Segment message acknowledgment received from remote NSP.
FLOWloc_dat	0	8	The normal data request count that will be sent in the next Data Request message.
FLOWloc_int	"empty"	2	The flow control state for receiving interrupt data. This variable takes into account the contents of the buffer BUFrcv as well as whether or not a new interrupt request should be sent. One of: "empty" "interrupt" "send request"
FLOWrem_dat	0	8	The cumulative normal data request count received from the remote NSP.
FLOWrem_int	1	8	The cumulative interrupt data request count received from the remote NSP.

Table 4 (Cont.)
Session Control Port

Name	Initial Value	Bit Width	Definition
FLOWrem_sw	"send"	1	The normal data on/off switch most recently received from the remote NSP. One of: "send" "do not send"
FLOWrem_typ	"none"	2	The flow control type of the remote receiver (determines how remote normal data request counts are interpreted). One of: "none" "segment" "session-control-message"
FLAGdat_ack	false	1	Boolean "data acknowledgment required" flag.
FLAGoth_ack	false	1	Boolean "other data acknowledgment required" flag.
FLAGbuf	false	1	Boolean "buffer required for connect or disconnect message" flag.
FLAGcon	false	1	Boolean "connect data available" flag.
FLAGint_avail	false	1	Boolean "transmit interrupt data available" flag.
FLAGcon_alloc	false	1	Boolean "transmit allocation requested for the port connect/ disconnect transmit process."
FLAGdat_alloc	false	1	Boolean "transmit allocation requested for the port data transmit process."
BUFxmt	0	8 bytes	Buffer to contain data for transmitted Connect Confirm, Disconnect Initiate, or Interrupt messages.
BUFrcv	0	8 bytes	Buffer to contain data for received Disconnect Initiate or Interrupt messages.
BUFacc	0	6 bytes	Buffer to contain accept data for received Connect Confirm message.

Table 4 (Cont.)
Session Control Port

Name	Initial Value	Bit Width	Definition
BUFcon	0's	*	Address of a buffer to contain data from a transmitted or received Connect Initiate message.
BUFsrc	0's	*	Address of a buffer to contain source node address for a received connect request.
COUNTretrans	0	16	Count of message retransmissions.
DELAYstr_tim	0	*	Round trip time-of-day value for start of round trip time estimation
DELAYmsg_num	0	12	The number of the Data Segment message currently being timed for the round trip delay estimation.
OTHERstate	"ready"	2	The state of the single "Other Data" message being transmitted, if any. One of: "ready" "sent" "timeout"
OTHERtyp	*	2	The type of "Other Data" message being sent, if any. It has meaning only when OTHERstate is not "ready". One of: "interrupt" "interrupt request" "data request"
CONFIDENCE	true	1	The Boolean "confidence" variable for the port.
SIZEseg	*	16	The transmit segment size.
TRYHARD	false	1	The Boolean "tryhard" variable for the port.

* This value is not essential to the model implementation.

5.3 Reserved Port Data Base

This data base contains a collection of port variables reserved for NSP's internal use. NSP uses them to manage responses to received messages that do not map onto the Session Control port data base. Table 5 describes a reserved port.

Table 5
Reserved Port

Name	Initial Value	Bit Width	Definition
NODE	0	16	Remote node address.
ADDRtmp	0	16	Temporary (local) link address.
ADDRrem	0	16	Remote link address.
MSGtype	"none"	2	The message type, if any that must be sent. One of: "no resources" "no link" "none"
CIRCUIT	0	*	The circuit number to use to transmit messages to Routing. One of: 0 (="unspecified") a circuit number
NEXTHOP	0	16	Node Number

5.4 Node Data Base

The node data base contains a collection of node descriptors. A node descriptor is a collection of node-dependent variables and counters. These are required in processing logical link connections. When NSP receives either an outgoing connect request (from Session Control) or an incoming connect request (via a Connect Initiate message), NSP attempts to allocate a node descriptor for the remote node if one does not exist. Failure of such an attempt has the same consequences as failure to allocate a port. Table 6 describes a node descriptor.

Table 6
Node Descriptor

Name	Initial Value	Bit Width	Definition
NODEaddr	*	16	Node address.
NODEdelay	0	*	Estimated round trip delay.
NODEactive	0	*	The number of ports with logical links to the remote node in a state other than OPEN.
NODEbyt_rcv	0	32	Total user data bytes received.
NODEbyt_xmt	0	32	Total user data bytes transmitted.
NODEmsg_rcv	0	32	Total NSP messages received.
NODEmsg_xmt	0	32	Total NSP messages transmitted.
NODEcon_rcv	0	16	Connect Initiate messages received.
NODEcon_xmt	0	16	Connect Initiate messages transmitted.
NODEcon_rej	0	16	Connect Initiate messages received for which there was no OPEN port.
NODEtimeout	0	16	Number of timeouts causing NSP message retransmission.

* This value is not essential to the model implementation.

5.5 Buffer Pools

There can be up to six buffer pools, as follows:

Large transmit buffer pool. Large transmit buffers are required to transmit Connect Initiate or Data Segment messages. The form these buffers take is implementation-dependent. Implementations that support buffer chaining may require only enough space in a large transmit buffer to build a message header. Other implementations may use buffers no larger than the size Routing can transmit. The minimum size for the pool is one buffer.

Small transmit buffer pool. Small transmit buffers are required to transmit messages other than Connect Initiate or Data Segment. The minimum size for this pool is one buffer. An implementation may use a single transmit buffer pool. In this case, the buffers must all be "large."

Receive buffer pool. Receive buffers are required to receive an NSP message from Routing. Minimum buffer size is 230 bytes. This size is equal to the contents of NSPbuf (Table 3, Section 5.1) plus 13 (the maximum length of a Data Segment header). All NSP messages are received into buffers of the same size. The minimum size for this pool is one buffer.

Commit buffer pool. Once data is placed in a commit buffer, the receiving node is committed to keeping it. Such data is acknowledged to the transmitter. A commit buffer is the same size as a receive buffer. This buffer pool is not required for the correct operation of NSP.

Cache buffer pool. Cache buffers hold received Data Segment messages that cannot be acknowledged either because they were received out of order or because there is no permanent storage (such as a commit buffer or a Session Control buffer) for them. A cache buffer is the same size as a receive buffer. This buffer pool is not required for the correct operation of NSP.

Event buffer pool. This contains buffers for NSP's event queue for reading by Network Management. The minimum size for the pool is one buffer.

6.0 DETAILED FUNCTIONAL MODEL

Section 6 is essentially a model implementation of NSP that defines the operation of NSP in terms of a high level language. There is no requirement that an actual implementation conform to either the structure or the logical flow of this model. This is a specification of function only. The following variables are used in the model:

- o Data base variables from Section 5. These are given in mixed capital and small letters plus STATE, NODE, VERSION, CONFIDENCE, CIRCUIT and NEXTHOP.
- o TIME. This refers to the value of a local clock that keeps the time of day.
- o Fields in a message from Section 8. These are all other variables and are given in capital letters.

All NSP segment number arithmetic in this section is performed modulo 4096. A segment number n is defined to be greater than a segment number m if 0 < (n - m) < 2048, modulo 4096.

A timer is modeled as a variable in a port. A timer is in one of three states: stopped, running, or expired. A timer contains a zero when it is stopped, a time value (the expiration time) greater than the time of day when it is running, and a time value less than or equal to the time of day when it has expired.

This section does not describe:

- The operation of the Network Management interface,
- The maintenance of counters in the node data base and the variable NSPmax in NSP's internal data base,
- The handling of NSP's event queue, or
- The handling of port pools.

It is assumed that the above operations are described sufficiently in the description of the Network Management interface in Section 3.

Finally, this section does not describe the operation of an NSP implementation that requests no flow control or message flow control. Nor does it describe the operation that would generate a negative acknowledgment or exercise "on/off" flow control. The operation of NSP in transmitting to an NSP implementation that uses these other forms of flow control and acknowledgment is described in Section 2.6.3.

The colloquial language in the model adheres to the following rules:

1. <-- is the assignment operator
2. < > means "not equal to"
3. <= means "less than or equal to"
4. >= means "greater than or equal to"
5. "Loop...Endloop" defines a section of logic that executes repeatedly. An "Exitloop" causes an exit to the logic immediately following the "Endloop."
6. "If...Elseif...Else...Endif" defines a collection of separate logic sections, each guarded by a Boolean condition. After the first section with a "true" Boolean guard is executed, an exit is made to the logic following the "Endif." The implied Boolean condition on "Else" is "true." That is, the section following an "Else" is always executed if a previous section has not been executed.
7. Comments are near the code to which they apply, either before or after.

6.1 Interface Routines

This section specifies how NSP handles the Session Control calls described in Section 3.1. The operation is described by a series of algorithms with comments. The algorithms assume that at each of the entry points involving a port identifier passed by Session Control, NSP does the following:

1. Checks the port identifier for validity.

2. Maps the port identifier onto a Session Control port descriptor, if possible.

The port descriptor variables used herein refer to those in the proper descriptor.

OPEN:

Logical link address assignment (Appendix A)

```
If (NSPstate = "halted") then
   return "no port allocated - NSP halted"
Elseif (Session Control port  is available
      and a logical link address is assignable) then
   allocate Session Control port
   ADDRloc <-- assigned link address
   STATE <-- "O"
   BUFcon <-- Session Control's buffer descriptor
   BUFsrc <-- address of "source" argument
   return "port allocated" with port identifier
Else
   return "port not allocated - insufficient resources"
Endif
```

CLOSE:

Logical link address deassignment (Appendix A)

```
deassign link address in ADDRloc
release resources
```

CONFIDENCE:

```
If (STATE = "RUN" or "CC" or "DR" or "DI") then
   return CONFIDENCE
Else
   return "port in invalid state"
Endif
```

STATE:

```
return STATE
```

CONNECT-XMT:

Logical link address assignment (Appendix A). Setting FLAGbuf true causes the connect/disconnect process for the port to send a Connect Initiate message. NSP passes the circuit,nexthop value to Routing for subsequent transmissions for this port. The circuit,nexthop value is used for loop-back testing.

```
If (NSPstate = "halted") then
   return "no port allocated - NSP halted"
Elseif (Session Control port  is available
      and a node descriptor exists or is available
      and a logical link address is assignable) then
   allocate Session Control port descriptor
   allocate and initialize a node descriptor, if necessary
   NODErem <-- destination from call
   ADDRloc <-- assigned link address
   STATE <-- "CI"
   BUFcon <-- Session Control's buffer descriptor
   CIRCUIT <-- circuit from call
   NEXTHOP <-- nexthop from call
   FLAGbuf <-- true
   return "port allocated" with port identifier
Else
   return "port not allocated - insufficient resources"
Endif
```

CONNECT-STATUS:

Accept or reject data is only available if the port is in the "RJ" or "RUN" states. The data is in BUFacc if the port is in the "RUN" state; the data is in BUFrcv if the port is in the "RJ" state. Furthermore, since BUFrcv receives interrupt data once the logical link is running, Session Control can obtain accept data only once. The variable FLOWloc_int identifies the contents of the BUFrcv buffer. Setting this variable to "empty" allows NSP to receive interrupt data on the logical link.

```
If (STATE = "RJ") then
   Session Control's buffer <-- BUFrcv
   return "connect request rejected"
Elseif (STATE = "RUN") then
   If (FLOWloc_int = "accept") then
       Session Control's buffer <-- BUFacc
   Endif
   return "connect request accepted"
Else
   return "port not in RUNNING or REJECTED state"
Endif
```

ACCEPT:

Setting FLAGbuf true will cause the connect/disconnect transmit process to send a Connect Confirm message.

```
If (STATE = "CR") then
   STATE <-- "CC"
   BUFxmt <-- Session Control's data
   FLAGbuf <-- true
   return "link accepted"
Else
   return "port not in CONNECT-RECEIVED state"
Endif
```

REJECT:

Setting FLAGbuf true causes the connect/disconnect transmit process to send a Disconnect Initiate message.

```
If (STATE = "CR") then
   STATE <-- "DR"
   BUFxmt <-- Session Control's data
   FLAGbuf <-- true
   return "link rejected"
Else
   return "port not in CONNECT-RECEIVED state"
Endif
```

DISCONNECT-XMT:

Setting FLAGbuf false and STATE to "DI" causes the connect/disconnect transmit process to send a Disconnect Initiate message only if there are no outstanding, unacknowledged data messages.

```
If (STATE = "RUN") then
      STATE <-- "DI"
      BUFxmt <-- Session Control's data
      DELAYstr_tim <-- 0
      FLAGbuf <-- false
      return "call accepted"
Else
   return "port not in RUNNING state"
Endif
```

ABORT-XMT:

This routine acts as routine DISCONNECT-XMT except that it sets NUMhigh to ACKrcv_dat to stop the transmission of data messages and to allow the transmission of a Disconnect Initiate message.

```
If (STATE = "RUN") then
   STATE <-- "DI"
   BUFxmt <-- session control's data
   DELAYstr_tim <-- 0
   stop timer TIMERdat
   stop timer TIMERoth
   FLAGbuf <-- false
   NUMhigh <-- ACKrcv_dat
   return "call accepted"
Else
   return "port not in RUNNING state"
Endif
```

DISCONNECT-RCV:

The connect/disconnect process places any received disconnect data in BUFrcv.

```
If (STATE = "DN") then
   If (BUFrcv empty) then
      return "no disconnect data available"
   Else
      Session Control's buffer <-- BUFrcv
      return "disconnect data available"
   Endif
Else
   return "port not in DISCONNECT-NOTIFICATION state"
Endif
```

DATA-XMT:

The segmentation module handles this call almost entirely.

```
If (STATE = "RUN") then
   pass call to segmentation module
   pass return to Session Control
Else
   return "port not in RUNNING state"
Endif
```

XMT-POLL:

The segmentation module handles this call entirely.

```
pass call to segmentation module
pass return to Session Control
```

DATA-RCV:

The reassembly module handles this call almost entirely. This call is accepted in the DI state (Section 4.1) to prevent a Session Control deadlock.

```
   If (STATE = "RUN", "DI" or "DN") then
      pass call to reassembly module
      pass return to Session Control
   Else
      return "port not in RUNNING or DISCONNECT-INITIATE state"
   Endif
```

RCV-POLL:

The reassembly module handles this call entirely.

```
   pass call to reassembly module
   pass return to Session Control
```

INTERRUPT-XMT:

BUFxmt is the single buffer holding outgoing interrupt data. The variable FLAGint_avail indicates the state of the buffer. When FLAGint_avail is false, then there is no data in it. Putting interrupt data in it and setting FLAGint_avail to true causes the data transmit process to send an Interrupt message.

```
   If (STATE < > RUN) then
      return "port not in RUNNING state"
   Elseif (FLAGint_avail false) then
      BUFxmt <-- Session Control's data
      FLAGint_avail <-- true
      return "data accepted"
   Else
      return "data not accepted - insufficient resources"
   Endif
```

INTERRUPT-RCV:

The data receive process places received interrupt data in BUFrcv if FLOWloc_int = "empty." The receive data process informs this routine that interrupt data is available by setting FLOWloc_int to "interrupt." This routine informs the data transmit process that it should request another interrupt message by setting FLOWloc_int to "send request." This causes the data transmit process to send an Interrupt Request message.

```
If (STATE < > RUN) then
   return "port not in RUNNING state"
Elseif (FLOWloc_int = "interrupt") then
   Session Control's buffer <-- BUFrcv
   FLOWloc_int <-- "send request"
   return "data returned"
Else
   return "no data returned"
Endif
```

6.2 Receive Dispatcher Module

This module has an imbedded process that gives receive buffers to Routing, polls to get them back, and then gives them to the receive processes. Although not explicitly modeled in the receive process algorithms, these processes return the receive buffers to the receive dispatcher module after the buffers have been processed.

The receive dispatcher module maps received messages onto ports and parses messages into field contents. Mapping a received message onto a particular port is described below.

1. If the TYPE or SUBTYPE subfields of the MSGFLG field contain reserved binary values, or if the MSGFLG field is extended, then discard the message.

2. Discard a received No Operation message.

3. A received Connect Initiate message or Retransmitted Connect Initiate message with DSTADDR = 0 maps onto any Session Control port with node = NODE, ADDRrem = SRCADDR and STATE <> "O" or any Session Control port with STATE = "O", if such a port exists and if a node descriptor exists or can be created for the source node. Otherwise, it maps onto any reserved port with MSGtyp = "none", if such a port exists. Otherwise, discard the message.

4. A returned Connect Initiate message or Retransmitted Connect Initiate maps onto the Session Control port with STATE = "CI" and SRCADDR=ADDRloc, if such a port exists. Otherwise, discard the message.

5. Treat a received Disconnect Confirm message as a No Resource message if the REASON field contains a 1, as a Disconnect Complete message if the REASON field contains a 42, and as a No Link message if the REASON field contains a 41.

6. The following messages map onto a Session Control port in the "CI" state with the source node = NODE and DSTADDR = ADDRloc, if such a port exists. These messages, with the exception of the Connect Acknowledgment message, also map onto a port in any other state with source node = NODE, DSTADDR = ADDRloc, and SRCADDR = ADDRrem, if such a port exists.

   - Connect Acknowledgment
   - No Resources
   - Connect Confirm
   - Disconnect Initiate
   - Disconnect Confirm with REASON < > 1, 41, or 42

   If a Connect Acknowledgment or No Resources message cannot be mapped onto a Session Control port, discard it. If a Connect Confirm or Disconnect Initiate message cannot be mapped onto a Session Control port, map it onto a reserved port with MSGtyp = "none," if one exists. Otherwise, discard it.

7. The following messages map onto a Session Control port with the source node=NODE, DSTADDR=ADDRloc, and SRCADDR=ADDRrem, if such a port exists.

   - Disconnect Complete
   - No Link
   - Data Segment
   - Data Acknowledgment
   - Interrupt
   - Data Request
   - Interrupt Request
   - Other-Data Acknowledgment

   A Data Segment, Interrupt, Data Request, or Interrupt Request message that cannot map onto a Session Control port is mapped onto a reserved port with MSGtyp = "none," if one exists; otherwise, it is discarded. Discard the remaining messages in the above list if they cannot map onto a Session Control port.

   Note that a Session Control port with STATE = "O," "CI," "CD," "NR," or "NC" does not have a defined value for ADDRrem. Therefore, NSP cannot map these messages onto such a port.

Parsing messages into field contents is generally straightforward. The following are special rules:

1. Check the QUAL fields of a received Data Segment, Interrupt, Link Service, Data Acknowledgment, or Other Acknowledgment message to determine if the fields are valid. Ignore a NUMBER field if the QUAL field has an invalid value.

2. Mask the SEGNUM field of a received Data Segment, Interrupt, Link Service, Data Acknowledgment, or Other Acknowledgment message to the low order 12 bits. Ignore the upper 4 bits regardless of setting.

3. Mask the LSFLAGS field of a received Link Service message to the low order 4 bits. Ignore the upper 4 bits regardless of setting. Check the reserved values of the FCVAL INT and FC MOD subfields of the LSFLAGS field, however. If the reserved values are used, ignore the entire Link Service message.

6.3 Index to Routines

Sections 6.4 through 6.9 contain routines that are used in more than one subsection. These routines are only defined once. To aid your reading of the model, Table 7 is an alphabetic list of all the routines used in these sections. The table shows both the section in which the routine is defined and the section(s) that call or otherwise use the routine.

Table 7
Index to Routines Used in Model

Routine	Defined In	Used In
ACK-SESSION-CONTROL	6.7	6.6.2
ALLOCATE	6.9	6.6.1, 2, 3
CHECK-ALLOCATE	6.9	6.6.1, 3
COMMIT-NUMBER	6.5	6.4.2
DATA-ACK-SENT	6.6.2	6.6.2
DATA-ACK-TO-BE-SENT	6.6.2	6.6.2
DATA-REQUEST-TO-BE-SENT	6.6.2	6.6.2
DATA-TO-BE-SENT	6.6.2	6.6.2
DEALLOCATE	6.9	6.6.1, 2, 3
GET-SEGMENT	6.8	6.6.2
INTERRUPT-TO-BE-SENT	6.6.2	6.6.2
INT-REQUEST-TO-BE-SENT	6.6.2	6.6.2
LAST	6.8	6.6.2
MESSAGE-SENT	6.6.2	6.6.2
MESSAGE-TO-BE-SENT	6.6.2	6.6.2
NM	6.8	6.4.2, 6.6.2
OTHER-ACK-SENT	6.6.2	6.6.2
OTHER-DATA-ACK-TO-BE-SENT	6.6.2	6.6.2
OTHER-DATA-SENT	6.6.2	6.6.2
PROCESS-DATA-ACK	6.4.2	6.4.2

Table 7 (Cont.)
Index to Routines Used in Model

Routine	Defined In	Used In
PROCESS-OTHER-DATA-ACK	6.4.2	6.4.2
REALLOCATE	6.9	6.6.1, 2, 3
SEND-CONNECT-ACKNOWLEDGMENT	6.7	6.6.1
SEND-CONNECT-CONFIRM	6.7	6.6.1
SEND-CONNECT-INITIATE	6.7	6.6.1
SEND-DATA-ACK	6.7	6.6.2
SEND-DATA-REQUEST	6.7	6.6.2
SEND-DATA-SEGMENT	6.7	6.6.2
SEND-DISCONNECT-COMPLETE	6.7	6.6.1
SEND-DISCONNECT-INITIATE	6.7	6.6.1
SEND-INTERRUPT	6.7	6.6.2
SEND-INTERRUPT-REQUEST	6.7	6.6.2
SEND-NO-LINK	6.7	6.6.3
SEND-NO-RESOURCES	6.7	6.6.3
SEND-OTHER-DATA-ACK	6.7	6.6.2
SEND-SMALL-MESSAGE	6.7	6.7
SET-SWITCH-AND-FLAG	6.4.2	6.4.2
SPECULATE-NUMBER	6.5	6.4.2
STORE-SEGMENT	6.5	6.4.2
TIMEOUT	6.6.2	6.6.1, 2
ROUTING-TRANSMIT	3.3	6.7
UPDATE-DELAY	6.4.2	6.4.1, 2

6.4 Receive Processes

This section contains algorithms for implementing the three receive processes:

- o Connect/Disconnect Receive Process (Section 6.4.1)
- o Data Receive Process (Section 6.4.2)
- o Reserved Receive Process (Section 6.4.3)

6.4.1 Connect/Disconnect Receive Processes - These processes receive messages from the receive dispatcher module. The message variables below refer to the information content of message fields as returned from the receive dispatcher module. There is one connect/disconnect receive process for each Session Control port.

```
Loop
```

This process loops forever.

```
   Oncase (received message type)
```

When this process receives a message, execute the appropriate case statement.

```
   Case (Connect Acknowledgment)
```

If this message is received when the link is in the "CI" state, then observe the round trip delay (Section 7.3). This value is equal to the current time of day minus the time the Connect Initiate message was sent. This latter time is kept in DELAYstr_tim. Routine UPDATE-DELAY makes this calculation and updates the variable NODEdelay.

```
      If (STATE = "CI") then
         STATE <-- "CD"
         Call UPDATE-DELAY
      Endif
```

```
   Case (Connect Initiate or Retransmitted Connect Initiate)
```

A Connect Initiate message may contain any value in the INFO field. However, the SERVICES field must contain "none," "segment" or "message". Setting FLAGbuf true causes the connect/disconnect transmit process to send a Connect Acknowledgment message.

```
    If (STATE = "O") then

         If (SERVICES = "none," "segment," or "message") then
            buffer whose descriptor is in BUFcon <-- DATA-CTL
            NODE <-- node from Routing
            location whose address is in BUFsrc <-- NODE
            ADDRrem <-- SRCADDR
            FLOWrem_typ <-- SERVICES
            VERSION <-- INFO
            SIZEseg <-- min  of (SEGSIZE, size of a large transmit
                        buffer minus the maximum NSP header size)
            FLAGbuf <-- true
            STATE <-- "CR"
            If (NODE = NSPself) then
               CIRCUIT <-- CIRCUIT received from Routing
               NEXTHOP <-- NEXTHOP received from Routing
            Else
               CIRCUIT <-- "unspecified"
            Endif
         Endif

   Elseif (STATE = "CR", "CC", or "DR") then FLAGbuf <-- true
   Elseif (STATE = "RUN", "CD", or "DI") then discard.

                    Message was a duplicate.

   Endif

Case ("returned to sender" Connect Initiate or Retransmitted
Connect Initiate)

   If (STATE = "CI") then STATE <-- "NC"
   Else discard
```

Case (Connect Confirm)

The testing of INFO and SERVICES is the same for a Connect Confirm message as for a Connect Initiate message. Setting the variable FLOWloc_int to "accept" informs the interface routines that there is accept data available. Since the "RUN" state can be entered, start the inactivity timer (Section 7.4). Call UPDATE-DELAY for the same reasons as when a Connect Acknowledgment message is received.

```
If (SERVICES = "none," "segment," or "message") then
   If (STATE = "CI" or "CD") then
      ADDRrem <-- SRCADDR
      FLOWrem_typ <-- SERVICES
      VERSION <-- INFO
      SIZEseg <-- min of (SEGSIZE, size of a large transmit
                  buffer -13)
      BUFacc <-- DATA-CTL
      TIMERinact <-- TIME + NSPinact_tim
      If (STATE = "CI") then Call UPDATE-DELAY
      STATE <-- "RUN"
   Endif
   If (STATE = "RUN") then
      FLAGdat_ack <-- true
   Endif
Endif
```

Case (Disconnect Initiate)

In all cases below, setting FLAGbuf true causes the connect/disconnect transmit process to send a Disconnect Complete message.

```
If (STATE = "CI" or "CD") then
```

A Disconnect Initiate message received in either of these two states indicates a rejection of a previously transmitted Connect Initiate message.

```
   ADDRrem <-- SRCADDR
   first two bytes of BUFrcv <-- REASON
   remaining bytes of BUFrcv <-- DATA-CTL
   FLAGbuf <-- true
   If (STATE = "CI") then Call UPDATE-DELAY
   STATE <-- "RJ"
Elseif (STATE = "RJ") then
```

A Disconnect Initiate message received in this state is assumed to be a duplicate caused by the retransmission of the message that originally caused the transition to the "RJ" state.

```
   FLAGbuf <-- true
Elseif (STATE = "RUN") then
```

A Disconnect Initiate message received in this state indicates a disconnect of a running link.

```
         first two bytes of BUFrcv <-- REASON
         remaining bytes of BUFrcv <-- DATA-CTL
         FLAGbuf <-- true
         STATE <-- "DN"
         DELAYstr_tim <-- 0
         stop timer TIMERdat
         stop timer TIMERoth
      Elseif (STATE = "DIC" or "DI") then
```

A Disconnect Initiate message received in either of these states is assumed to be a result of a disconnect collision. In the case of the "DIC" state, this message may have been delayed in Routing until after the reception of the Disconnect Complete message that caused the transition to the "DIC" state. In the case of the "DI" state, there has been a crossing of Disconnect Initiate messages in Routing. In either case, the response is the same.

```
         FLAGbuf <-- true
         STATE <-- "DIC"
         DELAYstr_tim <-- 0
         stop timer TIMERcon
      Elseif (STATE = "DN") then
```

A Disconnect Initiate message received in this state is assumed to be a duplicate caused by the retransmission of the message that originally caused the transition to the "DN" state.

```
         FLAGbuf <-- true
      Endif

   Case (No Resources)

      If (STATE = "CI") then
         STATE <-- "NR"
         Call UPDATE-DELAY
      Endif

   Case (Disconnect Complete)
```

Stopping timer TIMERcon prevents the retransmission of a Disconnect Initiate message after a timeout.

```
      If (STATE = "DR" or "DI") then
         stop timer TIMERcon
         stop timer TIMERdat
         stop timer TIMERoth
         Call UPDATE-DELAY
      Endif
      If (STATE = "DR") then STATE <-- "DRC"
      If (STATE = "DI") then STATE <-- "DIC"

   Case (No Link)
```

Stopping the timers prevents any retransmissions.

```
      If (STATE = "CC," "RUN," "DR" or "DI") then
         stop timer TIMERcon
         stop timer TIMERdat
         stop timer TIMERoth
         STATE <-- "CN"
      Endif

   Case (Disconnect Confirm)
```

This case is included only for compatibility with version 3.1. This case occurs when a 3.1 system sends this message rather than a Disconnect Initiate message for a Session Control rejection of a Connect Initiate message. It also occurs when an NSP version 3.1 system receives a message in error.

```
      If (STATE = "CI") then
         first two bytes of BUFrcv <-- REASON
         STATE <-- "RJ"
         CALL UPDATE_DELAY
      Elseif (STATE = "CC" or STATE = "RUN") then
         stop timer TIMERcon
         stop timer TIMERdat
         stop timer TIMEROth
         STATE <-- "CN"
      Endif
   Endcase
Endloop
```

6.4.2 Data Receive Processes - These processes receive messages from the receive dispatcher module. The message variables below refer to the information content of message fields as returned from the receive dispatcher module. There is one Session Control port data receive process for each Session Control port.

```
Loop
```

This process loops forever.

```
   If [(STATE = "CC" or "RUN")
         or (STATE = "DI" and NUMhigh < > ACKrcv_dat)] then
```

This process only runs for ports in the "CC," "RUN," or "DI" states. It runs for ports in the "CC" state to receive any data type message as defined by the case statements since the reception of these message types will cause a transition to the "RUN" state. It runs for ports in the "RUN" state to receive normal data, interrupts, Link Service messages, and acknowledgments. It runs for ports in the "DI" state to receive data while there is outstanding, unacknowledged, transmitted data. This latter case is true when NUMhigh < > ACKrcv_dat.

```
      If (FLOWloc_dat = 0) then
```

FLOWloc_dat contains the value, if any, currently being transmitted or retransmitted in a Data Request message. If it is zero, a new Data Request message can be sent. The routine SPECULATE-NUMBER returns the value to be sent in the next Data Request message.

```
         Call SPECULATE-NUMBER (low-overhead = "false")
         FLOWloc_dat <-- returned value
      Endif
```

The routine COMMIT-NUMBER returns the highest number of a previously received Data Segment that has been committed to either a commit buffer or a Session Control receive buffer.

```
      Call COMMIT-NUMBER

      If (returned value < > ACKxmt_dat) then
```

If the returned number is different from the last acknowledgment number sent, then a new acknowledgment must be sent. The variable ACKxmt_dat contains the data acknowledgment number to be sent in any Data Segment or Data Acknowledgment message transmitted. Setting FLAGxmt_ack true causes the data transmit process to send a Data Acknowledgment message if a Data Segment message does not need to be transmitted.

```
         FLAGdat_ack <-- true
         ACKxmt_dat <-- returned value
      Endif
   Endif
   If (message is received) then
      If (STATE = "CC" and message type = "Data Acknowledgment",
      "Interrupt", "Data Request", "Interrupt Request"
      or "Other-Data-Acknowledgment") then
```

If this message is taking the port out of the "CC" state, then make an estimate of the round trip delay to the remote NSP (Section 7.3). Since the port just entered the "RUN" state, start the inactivity timer (Section 7.4).

```
         STATE <-- "RUN"
         FLAGbuf <-- false
         Call UPDATE-DELAY
         stop timer TIMERcon
         CONFIDENCE <-- true
         COUNTretrans <-- 0
      Endif
```

Receiving any message restarts the inactivity timer.

```
      TIMERinact <-- TIME + NSPinact_tim
      Oncase (received message type)
```

When a message is received, execute the appropriate case statement.

Case (Data Segment)

Process the acknowledgment field of a received Data Segment message independently from the rest of the message.

```
If (ACKNUM field present) then
Call PROCESS-DATA-ACK
Endif
If (ACKOTH field present) then
Call PROCESS-OTHER-DATA-ACK
Endif
If (SEGNUM > ACKxmt_dat) then
   Call STORE-SEGMENT with DATA, EOM (from MSGFLG), and
   SEGNUM
Else
   FLAGdat_ack <-- true
Endif
```

Case (Interrupt)

Process the acknowledgment field of a received Interrupt message independently from the rest of the message.

```
If (ACKNUM field present) then
Call PROCESS-OTHER-DATA-ACK
Endif
If (ACKDAT field present) then
Call PROCESS-DATA-ACK
Endif
If (SEGNUM = ACKxmt_oth + 1 and FLOWloc_int = "empty") then
```

This model of NSP can only buffer one received Interrupt message at a time (Section 7.2). Its number must be one greater than the number of the last other-data message accepted (contained in variable ACKxmt_oth). When the variable FLOWloc_int contains "empty," there is room in buffer BUFrcv. Setting FLAGoth_ack true causes the data transmit process to acknowledge this Interrupt message.

```
   BUFrcv <-- DATA
   FLOWloc_int <-- "interrupt"
   ACKxmt_oth <-- ACKxmt_oth + 1
   FLAGoth_ack <-- true
Elseif (SEGNUM <= ACKxmt_oth) then
   FLAGoth_ack <-- true
Endif
```

```
      Case (Data Request)
```

Process the acknowledgment field of a received Data Request message independently from the rest of the message.

```
         If (ACKNUM field present) then
         Call PROCESS-OTHER-DATA-ACK
         Endif
         If (ACKDAT field present) then
         Call PROCESS-DATA-ACK
         Endif

         If (SEGNUM = ACKxmt_oth + 1) then
```

This model of NSP processes only one other-data message at a time (Section 7.2). It processes each one to completion. The number of the one it wants to process next is one greater than the one that it has most recently accepted and processed (and whose number is contained in ACKxmt_oth).

Basically, the algorithm for processing a Data Request message is to check its validity. Discard it if it is invalid. If it is valid,

(1) Store the data "send/do not send" switch in variable FLOWrem sw.

(2) Increment the acknowledgment number for the other-data channel.

(3) Set the flag FLAGoth_ack to true to force the data transmit process to acknowledge the receipt of this message. (Use the routine SET-SWITCH-AND-FLAG.)

(4) Update the variable FLOWrem_dat, if necessary, by adding the count from the Data Request message to it.

```
            If (FLOWrem_typ = "none") then
               Call SET-SWITCH-AND-FLAG
            Elseif (FLOWrem_typ = "segment") then
```

The following two statements define the validity check for a received Data Request message from a remote NSP that receives with "segment" flow control.

```
               If (-128 <= FCVAL <= 127) then
                  If (-128 <= FLOWrem_dat + FCVAL <= 127) then
                     FLOWrem_dat <-- FLOWrem_dat + FCVAL
                     Call SET-SWITCH-AND-FLAG
                  Endif
               Endif

            Elseif (FLOWrem_typ = "session-control-message") then
```

The following two statements define the validity check for a received Data Request message from a remote NSP that receives with "session-control message" flow control.

```
               If (0 <= FCVAL <= 127) then
```

```
                  If (0 < FLOWrem_dat + FCVAL <= 127) then
                     FLOWrem_dat <-- FLOWrem_dat + FCVAL
                     Call SET-SWITCH-AND-FLAG
                  Endif
               Endif
            Endif
         Elseif (SEGNUM <= ACKxmt_oth) then
            FLAG oth_ack <-- true
         Endif

      Case (Interrupt Request)
```

Process the acknowledgment field of a received Interrupt Request message independently from the rest of the message.

```
         If (ACKNUM field present) then
         Call PROCESS-OTHER-DATA-ACK
         Endif
         If (ACKDAT field present) then
         Call PROCESS-DATA-ACK
         Endif
         If (SEGNUM = ACKxmt_oth + 1) then
```

The message number check for received Interrupt Request messages is the same as for received Interrupt and Data Request messages. The following statement defines the validity check for a received Interrupt Request message.

```
            If (FCVAL >= 0 and 0 <= FLOWrem_int + FCVAL <= 127) then
               FLOWrem_int <-- FLOWrem_int + FCVAL
               ACKxmt_oth <-- ACKxmt_oth + 1
               FLAGoth_ack <-- true
            Endif
         Elseif (SEGNUM <= ACKxmt_oth) then
            FLAG oth_ack <-- true
         Endif

      Case (Data Acknowledgment)
         Call PROCESS-DATA-ACK
         If (ACKOTH field present) then
         CALL PROCESS-OTHER-DATA-ACK
         Endif

      Case (Non-Data Acknowledgment)
         Call PROCESS-OTHER-DATA-ACK
         If (ACKDAT field present) then
         Call PROCESS-DATA-ACK
         Endif

      Endcase
   Endif
Endloop
```

The data receive processes call the following routines:

PROCESS-DATA-ACK:

In the following routine, the variables NUMBER and QUAL are the contents of the fields of the same names from the received data segment, Data Acknowledgment Message or Data Request Message.

```
   If (ACKrcv_dat <= NUMBER <= NUMsent) then
```

If NUMBER is not greater than or equal to ACKrcv_dat, then this acknowledgment has been explicitly or implicitly processed before (NUMBER is processed when it is equal to ACKrcv_dat in case QUAL="nak" or QUAL="cross sub-channel nak"). If it is not less than or equal to NUMhigh, then it is acknowledging a message that was never sent. In either case, ignore the acknowledgment.

```
      TRYHARD <-- false
      CONFIDENCE <-- true
      COUNTretrans <-- 0
```

The following "If" block updates the flow control variable (FLOWrem_dat) that allows data transmission. Note that there are parallel constructions here:

(1) NM(ACKrcv_dat + 1, NUMBER) is the number of "end-of-message" data segments in the range (ACKrcv_dat + 1) to NUMBER, and

(2) (NUMBER - ACKrcv_dat) is the number of data segments in the same range.

```
      If (FLOWrem_typ = "segment") then
         FLOWrem_dat <-- FLOWrem_dat - (NUMBER - ACKrcv_dat)
      Elseif (FLOWrem_typ = "message")
         FLOWrem_dat <-- FLOWrem_dat - NM(ACKrcv_dat + 1, NUMBER)
      Endif
      If ((QUAL = "nak" or QUAL = "cross sub-channel nak") or NUMdat
      <= NUMBER) then
         NUMdat <-- NUMBER + 1
      Endif
```

If either this acknowledgment was a negative acknowledgment or the number of the next Data Segment message that was going to be sent (contained in NUMdat) is less than or equal to the number just acknowledged, then set the number of the next Data Segment message to transmit to the number just acknowledged plus one.

```
      ACKrcv_dat <-- NUMBER
```

The following code restarts the data retransmission timer only if there is remaining unacknowledged, transmitted data.

```
      stop timer TIMERdat
      If (ACKrcv_dat < NUMsent) then
         TIMERdat <-- TIME + (NODEdelay * NSPdelay)
      Endif
      If (DELAYmsg_num <= ACKrcv_dat and DELAYstr_tim < > 0) then
```

If DELAYstr_tim is non-zero, then a Data Segment message is being timed for an update to the round trip delay estimate (see Section 7.3). If the Data Segment message being timed (whose number is contained in DELAYmsg_num) has just been acknowledged, then calculate the round trip delay for that message.

```
         Call UPDATE-DELAY
      Endif
   Endif
```

PROCESS-OTHER-DATA-ACK:

In the following routine, the variables NUMBER and QUAL are the contents of the fields of the same name from the received Interrupt, Data Request, Interrupt Request, Other-Data-Acknowledgment message, or Data Segment message.

```
   If (NUMBER=NUMoth-1 and (QUAL="nak" or QUAL="cross sub-channel
   nak")) then
      OTHERstate <-- "timeout"
   Endif
   If (NUMBER = NUMoth and OTHERstate < > "ready") then
```

NUMoth contains the number of the single, outstanding other-data message, if any (Section 7.2). OTHERstate is equal to either "sent" or "timeout" (i.e., not equal to "ready") if there is such an outstanding message. If the message is acknowledged, stop the retransmission timer for the message.

```
      stop timer TIMERoth
      TRYHARD <-- false
      CONFIDENCE <-- true
      COUNTretrans <-- 0
```

Handle a negative acknowledgment in the same manner as a timeout.

If the message was positively acknowledged, then send a new other-data message.

Setting OTHERstate to "ready" allows the data transmit process to send another other-data message. NUMoth contains the number of the next other-data message that will be transmitted.

```
         OTHERstate <-- "ready"
         NUMoth <-- NUMoth + 1
         If (OTHERtyp = "interrupt") then
```

OTHERtyp contains the type of other-data message that was sent and has just been acknowledged. If it was an Interrupt message, then a new one may be buffered into BUFxmt. Indicate this condition by setting FLAGint_avail to false. Decrement the remote interrupt request count.

```
            FLAGint-avail <-- false
            FLOWrem_int <-- FLOWrem_int - 1
         Elseif (OTHERtyp = "data request") then
```

If a Data Request message has just been acknowledged, then clear the count value contained in it, allowing the data receive process to obtain a new speculate number that can be sent in the next Data Request message.

```
            FLOWloc_dat <-- 0
         Endif
      Endif
```

The acknowledgment of an Interrupt Request message does not require any additional explicit handling. This is because a new Interrupt Request message cannot be sent until session control has received the interrupt data whose request was just acknowledged (Section 7.2).

SET-SWITCH-AND-FLAG:

This routine stores the data "send/do not send" value from a received Data Request message in FLOWrem_sw and sets FLAGoth_ack true to indicate that an other-data acknowledgment is required.

```
   FLOWrem_sw <-- FC MOD
   ACKxmt_oth <-- ACKxmt_oth + 1
   FLAGoth_ack <-- true
```

Both the data receive processes and the connect/disconnect receive processes call the following routine:

UPDATE-DELAY:

This routine updates the NODEdelay variable in a node descriptor. Call the routine with a port argument. In this code, "temp" is a temporary variable.

```
   If(DELAYstr_tim <> 0)then
      If(NODEdelay = 0)then
         NODEdelay <-- TIME - DELAYstr_tim
      Else
         temp <-- TIME - DELAYstr_tim
         temp <-- temp - NODEdelay
         NODEdelay <-- NODEdelay + (temp / (NSPweight + 1))
      Endif
      DELAYstr_tim <-- 0
   Endif
```

6.4.3 Reserved Receive Processes - These processes handle all received messages that map onto a reserved port. There is one reserved port process for each reserved port.

Loop

This process loops forever.

```
   If (a message is received) then
      If (the message is a Connect Initiate) then
```

Setting MSGtyp to "no resources" causes the reserved port transmit process to send a No Resources message.

```
         NODE <-- source node address
         ADDRrem <-- SRCADDR
         MSGtyp <-- "no resources"
         CIRCUIT <-- CIRCUIT received from Routing
         NEXTHOP <-- NEXTHOP recieved from Routing

      Elseif (the message is a Connect Confirm, Disconnect Initiate
              Data Segment, Interrupt, Data Request, or Interrupt
              Request) then
```

Setting MSTtyp to "no-link" causes the reserved port transmit process to send a No-Link message.

```
         NODE <-- source node address
         ADDRrem <-- SRCADDR
         ADDRtmp <-- DSTADDR
         MSGtyp <-- "no-link"
         CIRCUIT <-- circuit received from Routing
         NEXTHOP <-- nexthop received from Routing
      Endif
   Endif
Endloop
```

6.5 Reassembly Module

The reassembly module has two functions. First, it maps data from Data Segment messages into Session Control buffers according to the rules implied by the Session Control interface description. Because of this, the DATA-RCV call is passed to this module from the interface routines. Second, it manages the cache and commit buffer pools. This function includes flow control policy establishment. That is, this module is aware of the amount of buffering available via Session Control receive buffers, cache buffers, and commit buffers. It uses this information to produce normal data request counts to be sent in Data Request messages. The data transmit processes execute the flow control mechanism (in other words, the Data Request message transmission).

The detailed description of the operation of this module is beyond the scope of this specification. However, it must operate with the following restriction with respect to the management of the cache buffer pool. It must not discard the data from a received Data Segment message for a given Session Control port if there is data from a higher numbered Data Segment message in the cache for the same port. Appendix C contains an example of this module.

The data receive processes obtain changes in the number of Data Segment messages that should be received for a given port by calling this module. The data receive processes store these values in the corresponding ports. In the most general case, the reassembly module may be requesting data for which there is not necessarily any guaranteed storage. Therefore, the number of segments that the reassembly module wants to receive for a port at any given time is referred to as the "speculate number" for the port.

When the call to "SPECULATE-NUMBER" returns a non-zero value NSP will transmit a Data-Request message. The decision to send this message represents a tradeoff between the benefit of precise control over the flow of arriving messages and the cost of sending Data-request messages and their acknowledgments. The cost of sending a Data-Request message depends on whether the Data-Request message can piggyback a data acknowledgment. In order for the reassembly module to make intelligent policy decisions a parameter is defined, low-overhead, which is used to inform the reassembly module whether sending a Data-Request message at this time will be inexpensive. The data receive processes obtain the speculate number for a port by issuing the following call:

<u>SPECULATE-NUMBER</u> (port id, low-overhead ; return)

port id: a port identifier

low-overhead: low-overhead flag

return: the current number of data segments that this module would like to receive.

The data receive processes periodically obtain the highest segment number committed to either a commit or session control buffer. This value is the acknowledgment number to be sent to the remote NSP module. The data receive processes obtain this number by issuing the following call:

<u>COMMIT-NUMBER</u> (port id; return)

port id: a port identifier

return: the highest segment number committed

The data receive processes attempt to put data from received Data Segment messages into a cache, commit, or session control receive buffer by issuing the following call:

STORE-SEGMENT (port id, data, eom, number)

port id: a port identifier

data: data from a Data Segment message

eom: one of:

- o data is end-of-message
- o data is not end-of-message

number: the segment number of the Data Segment message

6.6 Transmit Processes

This section contains algorithms for implementing the following:

- o Connect/disconnect transmit process (Section 6.6.1)
- o Data transmit processes (Section 6.6.2)
- o Reserved transmit processes (Section 6.6.3)

6.6.1 Connect/Disconnect Transmit Processes

- These processes transmit Connect Initiate, Connect Acknowledgment, Connect Confirm, Disconnect Initiate, and Disconnect Complete messages. There is one session control port connect/disconnect transmit process for each Session Control port.

Loop

This process loops forever. In general, if FLAGbuf is true a connect or disconnect message requires transmission.

```
   If (STATE = "DI" and FLAGbuf false
         and NUMhigh = ACKrcv_dat and TIMERcon not running) then
```

This test causes this process to attempt to send a Disconnect Initiate message only if:

(1) Session Control requested a disconnection (i.e., STATE = "DI");

(2) there are no outstanding, unacknowledged Data Segment messages (NUMhigh = ACKrcv_dat); and

(3) a previously transmitted Disconnect Initiate message, if any, is not being timed out (TIMERcon not running).

```
      FLAGbuf <-- true
      DELAYstr_tim <-- TIME
   Endif
   If (timer TIMERcon expired and STATE = "CI", "CC", "DI", or "DR")
   then
```

If the retransmission timer has expired, NSP increments the retransmission count and attempts to send a connect or disconnect message.

```
      FLAGbuf <-- true
      Call TIMEOUT
   Endif
```

When FLAGbuf is true, a connect or disconnect message requires transmission. When FLAGcon_alloc is true, permission to transmit has been requested from the transmit allocation module. These two variables act together to form a state variable with 4 states. Conditions that require a message to be sent can change dynamically. This process has to request permission to transmit but has to take back a request it previously made if it no longer must send a message. Therefore, interpret these variables as follows:

FLAGbuf true, FLAGcon_alloc false: request permission to send.
FLAGbuf true, FLAGcon_alloc true: attempt message transmission.
FLAGbuf false, FLAGcon_alloc true: take back permission request.
FLAGbuf false, FLAGcon_alloc false: do nothing.

```
   If (FLAGbuf true and FLAGcon_alloc false) then
      Call ALLOCATE
      FLAGcon_alloc <-- true
   Elseif (FLAGbuf false and FLAGcon_alloc true) then
      Call DEALLOCATE
      FLAGcon_alloc <-- false
   Elseif (FLAGbuf true and FLAGcon_alloc true) then
      If (CHECK-ALLOCATE) then
```

CHECK-ALLOCATE is a Boolean function in the transmit allocation module that returns true if and only if permission to transmit has been granted. The state of the port determines the message to be sent.

```
         If (STATE = "CI") then
            DELAYstr_tim <-- TIME
            Call SEND-CONNECT-INITIATE with data addressed in BUFcon
         Endif
         If (STATE = "DIC," "RJ," or "DN") then
            Call SEND-DISCONNECT-COMPLETE
         Endif
         If (STATE = "DR" or "DI") then
            If (DELAYstr_tim = 0) then DELAYstr_tim <-- TIME
            Call SEND-DISCONNECT-INITIATE
         Endif
         If (STATE = "CR") then Call SEND-CONNECT-ACKNOWLEDGMENT
         If (STATE = "CC") then
            If (DELAYstr_tim = 0) then DELAYstr_tim <-- TIME
            Call SEND-CONNECT-CONFIRM
         Endif
```

If the transmission was successful, this process must take back its request for permission to transmit (to allow another process to be given an equal chance to transmit).

```
         If (success) then
            Call DEALLOCATE
            FLAGcon_alloc <-- false
            FLAGbuf <-- false
            If (STATE = "CC," "DI," or "DR") then
               If not (STATE = "CC" and VERSION = 3.1) then
               If (NODEdelay = 0) then
```

This means that there is no current round trip delay estimate to the remote node. The 5 seconds is a suggested value; it is not mandatory.

```
                  TIMERcon <-- TIME + 5 seconds
               Else
```

An estimate does exist.

```
                  TIMERcon <-- TIME + (NODEdelay * NSPdelay)
               Endif
            Endif
         Endif
         If (STATE = "CC" and VERSION = 3.1) then
            STATE <-- "RUN"
            stop timer TIMERcon
            CONFIDENCE <--true
            COUNTretrans <-- 0
            TIMERinact <-- TIME + NSPinact_tim
         Endif
      Else
```

If transmission is unsuccessful, calling REALLOCATE will give other ports a chance to transmit without removing this port for contention for Routing resources. This process leaves FLAGbuf true. This causes the process to request permission to transmit again.

```
            Call REALLOCATE
         Endif
      Endif
   Endif
Endloop
```

6.6.2 Data Transmit Processes - These processes send Data Segment, Interrupt, Data Request, Interrupt Request, Data Acknowledgment, and Other-Data Acknowledgment messages. There is one data transmit process for each Session Control port.

```
Loop
```

This process loops forever.
The data receive process for the port puts the highest number of an acknowledged Data Segment message in ACKrcv_dat. This process informs the segmentation module of this value by calling ACK-SESSION-CONTROL. It obtains the highest segment number available from the segmentation routines via function LAST.

```
   Call ACK-SESSION-CONTROL with ACKrcv_dat
   If (STATE = "RUN") then NUMhigh <-- LAST
   If [STATE = "RUN" or (STATE = "DI" and  NUMhigh  <  >  ACKrcv_dat)]
   then
```

With the exception of the processing above, this process does nothing for a port that is not in either the "RUN" state or in the "DI" state with outstanding, unacknowledged Data Segment messages (NUMhigh < > ACKrcv_dat).

```
      If (timer TIMERdat expired) then
```

If the data retransmission timer expires, then the number of the next Data Segment message to transmit is one greater than the highest segment acknowledged by the remote NSP.

```
         NUMdat <-- ACKrcv_dat + 1
         Call TIMEOUT
      Endif
      If (timer TIMERoth expired) then
```

If the other-data retransmission timer expires, record the expiration in the state variable OTHERstate. This is possible since there can be at most one outstanding, unacknowledged other data message in this model of NSP.

```
         OTHERstate <-- "timeout"
         Call TIMEOUT
      Endif
```

The tests below are completely analogous to the tests performed in a connect/disconnect transmit process. The only difference is that the need to transmit a connect or disconnect message could be determined by examining a single flag (FLAGbuf). But the need to send a data type message is determined by a complicated Boolean function of variables in the port. The Boolean function MESSAGE-TO-BE-SENT returns true if a message requires transmission and is used below in a manner analogous to FLAGbuf in a connect/disconnect transmit process.

```
      If (MESSAGE-TO-BE-SENT and FLAGdat_alloc false) then
         Call ALLOCATE
         FLAGdat_alloc <-- true
      Elseif (not MESSAGE-TO-BE-SENT and FLAGdat_alloc true) then
         Call DEALLOCATE
         FLAGdat_alloc <-- false
      Elseif (MESSAGE-TO-BE-SENT and FLAGdat_alloc true) then
         If (CHECK-ALLOCATE) then
```

If permission to transmit has been granted, then determine the type of message to transmit by testing a collection of Boolean functions of variables in the port. The order in which these functions are tested is important and is part of the specification. That is, if more than one type of data message may be transmitted, the type that must be transmitted first is important. Also, if transmission is unsuccessful, call REALLOCATE for the same reasons as in the connect/disconnect transmit processes.

```
            If (INTERRUPT-TO-BE-SENT) then
               Call SEND-INTERRUPT
               If (success) then
```

The routine OTHER-DATA-SENT performs processing common to the successful transmission of Interrupt, Data Request, and Interrupt Request messages.

```
                  OTHERtyp <-- "interrupt"
                  Call OTHER-DATA-SENT
               Else
                  Call REALLOCATE
               Endif
            Elseif (INT-REQUEST-TO-BE SENT) then
               Call SEND-INTERRUPT-REQUEST
               If (success) then
                  OTHERtyp <-- "interrupt request"
```

Setting FLOWloc_int to "empty" allows data from a received Interrupt message to be saved in BUFrcv by the data receive process.

```
                  FLOWloc_int <-- "empty"
                  Call OTHER-DATA-SENT
               Else
                  Call REALLOCATE
               Endif
            Elseif (DATA-REQUEST-TO-BE-SENT) then
               Call SEND-DATA-REQUEST
               If (success) then
                  OTHERtyp <-- "data request"
```

One reason that a Data Request message may be sent is that the inactivity timer has expired (Section 7.4). If this is the case, the inactivity timer is restarted.

```
                  If (TIMERinact expired) then
                     TIMERinact <-- TIME + NSPinact_tim
                  Endif
                  Call OTHER-DATA-SENT
                  If (VERSION = "4.0") then
                     FLAGdat_ack <-- false
                  Endif
               Else
                  Call REALLOCATE
               Endif
            Elseif (DATA-TO-BE-SENT) then
```

If there is no Data Segment currently being timed for an update to the estimated round trip delay (DELAYstr_tim = 0), then save the current time of day. Also save the number of the Data Segment being timed.

```
               If (DELAYstr_tim =0) then
                  DELAYstr_tim <-- TIME
                  DELAYmsg_num <-- NUMdat
               Endif
               Call GET-SEGMENT with segment number NUMdat
               Call SEND-DATA-SEGMENT with returned values
               If (success) then
```

The routine DATA-ACK-SENT performs processing common to the successful transmission of Data Segment and Data Acknowledgment messages.

If the Data Segment just sent had a number one greater than the highest number acknowledged by the remote NSP, then start the retransmission timer. The value for this timer is a constant times the current estimated round trip delay (Section 7.3).

```
                  If (NUMdat = ACKrcv_dat+1) then
                     TIMERdat <-- TIME + (NODEdelay * NSPdelay)
                  Endif
```

Increment the number of the next Data Segment to be transmitted (NUMdat).

```
                  If (NUMdat > NUMsent) then
                     NUMsent = NUMdat
                  Endif
                  NUMdat <-- NUMdat + 1
                  Call DATA-ACK-SENT
                  If (VERSION = "4.0") then
                     FLAGoth_ack <-- false
                  Endif
               Else
                  Call REALLOCATE
               Endif
            Elseif (OTHER-DATA-ACK-TO-BE-SENT) then
               Call SEND-OTHER-DATA-ACK
               If (success) then
```

The routine OTHER-ACK-SENT performs processing common to the successful transmission of Interrupt, Data Request, Interrupt Request, and Other-Data Acknowledgment messages.

```
                  Call OTHER-ACK-SENT
               Else
                  Call REALLOCATE
               Endif
         Elseif (DATA-ACK-TO-BE-SENT) then
            If (VERSION = "4.0") then
            Call SPECULATE-NUMBER (low-overhead = "true")
               FLOWloc_dat <-- returned value
               If (DATA-REQUEST-TO-BE-SENT) then
                  Call SEND-DATA-REQUEST
                  If (success) then
                     OTHERtyp <-- "data request"
```

One reason that a Data Request message may be sent is that the inactivity timer has expired (Section 7.4). If this is the case, the inactivity timer is restarted.

```
                     If (TIMERinact expired) then
                      TIMERinact <-- TIME + NSPinact_tim
                     Endif
                     Call OTHER-DATA-SENT
                  Else
                     Call REALLOCATE
                  Endif
               Else
                     Call SEND-DATA-ACK
                     If (success) then
                        Call DATA-ACK-SENT
                     Else
                        Call REALLOCATE
                     Endif
               Endif
            Else
               Call SEND-DATA-ACK
               If (success) then
                  Call DATA-ACK-SENT
               Else
                  Call REALLOCATE
               Endif
            Endif
         Endif
      Endif
   Endif
Endloop
```

The following routines are used by these processes.

TIMEOUT:

```
COUNTretrans <-- COUNTretrans + 1
If (COUNTretrans > NSPretrans) then CONFIDENCE <-- false
If (COUNTretrans > NSPtryhard) then TRYHARD <-- true
```

MESSAGE-TO-BE-SENT:

```
If (INTERRUPT-TO-BE-SENT) then return true
Elseif (INT-REQUEST-TO-BE-SENT) then return true
Elseif (DATA-REQUEST-TO-BE-SENT) then return true
Elseif (DATA-TO-BE-SENT) then return true
Elseif (OTHER-DATA-ACK-TO-BE-SENT) then return true
Elseif (DATA-ACK-TO-BE-SENT) then return true
Else
   return false
Endif
```

INTERRUPT-TO-BE-SENT:

To send an interrupt message, either:

(1) Interrupt data must be available in BUFxmt (FLAGint_avail true),
(2) The remote NSP must have requested the interrupt data (FLOWrem_int > 0), and
(3) There must be no outstanding, unacknowledged Interrupt, Data Request, or Interrupt Request message (OTHERstate = "ready").

or

(1) There must be an outstanding, unacknowledged Interrupt message (OTHERtyp = "interrupt"), and
(2) The message must have timed out (OTHERstate = "timeout").

```
If (FLAGint_avail true and FLOWrem_int > 0
    and OTHERstate = "ready") then
   return true
Elseif (OTHERstate = "timeout"
       and OTHERtyp = "interrupt") then
   return true
Else
   return false
Endif
```

INT-REQUEST-TO-BE-SENT:

To send an Interrupt Request message, either:

(1) The buffer BUFrcv must be empty and an Interrupt Request message not previously sent (FLOWloc_int = "send request"); and
(2) There must be no outstanding, unacknowledged Interrupt, Data Request, or Interrupt Request message (OTHERstate = "ready"),

or

(1) There must be an outstanding, unacknowledged Interrupt Request message (OTHERtyp = "interrupt request"); and
(2) The message must have timed out (OTHERstate = "timeout").

```
If (FLOWloc_int = "send request" and OTHERstate = "ready") then
   return true
Elseif   (OTHERstate   =   "timeout" and  OTHERtyp = "interrupt
   request," then
   return true
Else
   return false
Endif
```

DATA-REQUEST-TO-BE-SENT:

To send a Data Request message, either:

(1) There must be an unsent request count (FLOWloc_dat < > 0); and
(2) There must be no outstanding, unacknowledged Interrupt, Data Request, or Interrupt Request message (OTHERstate = "ready").

or

(1) The activity timer must have expired (TIMERact expired); and
(2) There must be no outstanding, unacknowledged Interrupt, Data Request, or Interrupt Request message (OTHERstate = "ready").

or

(1) There must be an outstanding, unacknowledged Data Request message (OTHERtyp = "data request"); and
(2) The message must have timed out (OTHERstate = "timeout").

```
If (FLOWloc_dat < > 0 and OTHERstate= "ready") then
   return true
Elseif (TIMERinact expired and OTHERstate = "ready") then
   return true
Elseif (OTHERstate = "timeout"
         and OTHERtyp = "data request") then
   return true
Else
   return false
Endif
```

OTHER-DATA-ACK-TO-BE-SENT:

```
return FLAGoth_ack
```

DATA-TO-BE-SENT:

```
If (NUMdat <= NUMhigh and FLOWrem_sw = "send") then
```

To consider sending the next Data Segment to be transmitted (the one with number NUMdat), the Data Segment must be available from Session Control (NUMdat <= NUMhigh) and the remote NSP must be allowing data to be sent (FLOWrem_sw = "send"). Note that the next Data Segment message to be sent is always one that is unacknowledged since the variable NUMdat is always greater than ACKrcv_dat, although there will be no data to send if NUMdat = ACKrcv_dat + 1 = NUMhigh + 1.

The remaining tests depend on the type of flow control that was selected by the remote NSP when the logical link was established.

```
      If (FLOWrem_typ = "none") then
         return true
```

The two tests below are analogous. Each is testing to see if the number of requested elements (segments or Session Control messages) is greater than or equal to the number of elements in the range from the highest acknowledged (ACKrcv_dat) to the one whose transmission is being attempted (NUMdat).

```
      Elseif (FLOWrem_typ = "segment"
               and NUMdat <= ACKrcv_dat + FLOWrem_dat) then
         return true
      Elseif (FLOWrem_typ = "session-control-message"
               and NM(ACKrcv_dat + 1,NUMdat) <= FLOWrem_dat) then
         return true
      Else
         return false
      Endif
   Else
      return false
   Endif
```

DATA-ACK-TO-BE-SENT:

```
   return FLAGdat_ack
```

OTHER-DATA-SENT:

```
   Call OTHER-ACK-SENT
   OTHERstate <-- "sent"
   TIMERoth <-- TIME + (NODEdelay * NSPdelay)
```

OTHER-ACK-SENT:

```
   Call MESSAGE-SENT
   FLAGoth_ack <-- false
```

DATA-ACK-SENT:

```
   Call MESSAGE-SENT
   FLAGdat_ack <-- false
```

MESSAGE-SENT:

This routine ascertains if there is more data to be sent. If so, it calls REALLOCATE to remain in contention for transmit resources but to free those resources for other ports. If not, it calls DEALLOCATE to free the resources and remove itself from contention. In the latter case, it sets FLAGdat_alloc false so that an ALLOCATE request will be made when there is data to send.

```
   If (MESSAGE-TO-BE-SENT) then
      Call REALLOCATE
   Else
      Call DEALLOCATE
      FLAGdat_alloc <-- false
   Endif
```

6.6.3 Reserved Transmit Processes - The reserved transmit processes send No Resources and No-Link messages. There is one reserved transmit process for each reserved port.

Loop

The processing here is somewhat complicated due to the interaction with the transmit allocation module and the fact that a transmit request to Routing may fail. It is not modeled analogously to the connect/disconnect and data transmit processes because the need to transmit a given message will not disappear as with the other processes.

```
   If (MSGtyp < > "none") then
      Call ALLOCATE
      Loop
         Call CHECK-ALLOCATE
         If (success) then
            If (MSGtyp = "no resources") then
               Call SEND-NO-RESOURCES
            Elseif (MSGtyp = "no-link") then
               Call SEND-NO-LINK
            Endif
            If (success)
               MSGtyp <-- "none"
               Call DEALLOCATE
               Exitloop
            Else
               Call REALLOCATE
            Endif
         Endif
      Endloop
   Endif
Endloop
```

6.7 Transmit Format Module

This module manages the large and small transmit buffer pools, formats outgoing NSP messages, and sends messages to Routing. In addition, although it is not explicitly modeled, this module polls Routing to get back buffers that have been transmitted. When such a buffer is returned, it is immediately placed back in its pool.

The name of each routine in this module describes its function. Each call supplies the appropriate port id.

SEND-CONNECT-INITIATE (port id):

```
If (a large transmit buffer is available) then
   allocate large transmit buffer
   If (COUNTretrans = 0) the MSGFLG <-- "connect initiate"
   Else
      MSGFLG = "retransmit connect initiate"
   Endif
   SRCADDR <-- ADDRloc
   SERVICES <-- "segment"
   INFO <-- "version 4.0"
   SEGSIZE <-- NSPbuf
   DATA-CTL <-- data addressed by BUFcon
   Call ROUTING-TRANSMIT with "return to sender" and circuit =
   CIRCUIT,
   nexthop = NEXTHOP
   If (success) then
      return success
   Else
      release large transmit buffer
      return failure
   Endif
Else
   return failure
Endif
```

SEND-CONNECT-ACKNOWLEDGMENT (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "connect acknowledgment"
   DSTADDR <-- ADDRrem
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-NO-RESOURCES (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "disconnect confirm"
   DSTADDR <-- ADDRrem
   SRCADDR <-- 0
   REASON <-- 1
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-CONNECT-CONFIRM (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "connect confirm"
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRloc
   SERVICES <-- "segment"
   INFO <-- "version 4.0"
   SEGSIZE <-- NSPbuf
   DATA-CTL <-- BUFxmt
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-DISCONNECT-INITIATE (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "disconnect initiate"
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRloc
   REASON <-- first two bytes of BUFxmt
   DATA-CTL <-- remaining bytes of BUFxmt
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-DISCONNECT-COMPLETE (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "disconnect confirm"
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRloc
   REASON <--  42
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-NO-LINK (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "disconnect confirm"
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRtmp
   REASON <--  41
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-DATA-ACK (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "data acknowledgment"
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRloc
   NUMBER <-- ACKxmt_dat
   QUAL <-- "ack"
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-OTHER-DATA-ACK (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "other data acknowledgment"
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRloc
   NUMBER <-- ACKxmt_oth
   QUAL <-- "ack"
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-DATA-SEGMENT (port id, buffer descriptor, eom, bom):

```
If (a large transmit buffer is available) then
   allocate large transmit buffer
   MSGFLG <-- "data," eom, bom
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRloc
   NUMBER <-- ACKxmt_dat
   QUAL <-- "ack"
   NUMBER <-- ACKxmt_oth
   QUAL <-- "cross sub-channel ack"
   SEGNUM <-- NUMdat
   DATA <-- data from buffer descriptor
   Call ROUTING-TRANSMIT with circuit = CIRCUIT,
   nexthop = NEXTHOP and tryhard = TRYHARD
   If (success) then
      return success
   Else
      release large transmit buffer
      return failure
   Endif
Else
   return failure
Endif
```

SEND-INTERRUPT (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "interrupt"
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRloc
   NUMBER <-- ACKxmt_oth
   QUAL <-- "ack"
   SEGNUM <-- NUMoth
   DATA <-- BUFxmt
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-DATA-REQUEST (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "data request"
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRloc
   NUMBER <-- ACKxmt_oth
   QUAL <-- "ack"
   NUMBER <-- ACKxmt_dat
   QUAL <-- "cross sub-channel ack"
   SEGNUM <-- NUMoth
   FC MOD <-- "send"
   FCVAL INT <-- "data"
   FCVAL <-- FLOWloc_dat
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

SEND-INTERRUPT-REQUEST (port id):

```
If (a small transmit buffer is available) then
   allocate small transmit buffer
   MSGFLG <-- "interrupt request"
   DSTADDR <-- ADDRrem
   SRCADDR <-- ADDRloc
   NUMBER <-- ACKxmt_oth
   QUAL <-- "ack"
   SEGNUM <-- NUMoth
   FC MOD <-- "send"
   FCVAL INT <-- "interrupt"
   FCVAL <-- 1
   Call SEND-SMALL-MESSAGE
Else
   return failure
Endif
```

The following routine is used by the above routines.

SEND-SMALL-MESSAGE:

```
Call ROUTING-TRANSMIT with circuit = CIRCUIT, nexthop = NEXTHOP
   and tryhard = TRYHARD
If (success) then
   return success
Else
   release small transmit buffer
   return failure
Endif
```

6.8 Segmentation Module

The segmentation module maps data from Session Control transmit buffers into Data Segment messages according to the rules implied by the Session Control interface specification. The module makes the data available to the data transmit processes. Because of this, the interface routines pass the DATA-XMT and XMT-POLL calls to the module.

The detailed specification of this module is beyond the scope of this specification (Appendix B).

A data transmit process obtains a buffer descriptor for a segment of a session control message from this module by issuing the following call:

GET-SEGMENT (port id, number; return)

number: the number of the desired segment

returns: a buffer descriptor for the segment, an end-of-message indication, and a beginning-of-message indication

A data transmit process informs this module that it will no longer require a segment or sequence of segments by issuing the following call.

ACK-SESSION-CONTROL (port id, number)

number: a segment number; no segment with this or a lower number will be required again by the data transmit process.

A data transmit process obtains the number of Session Control data segments marked as "end-of-message" in a given range of segment numbers by executing the following function:

NM (port id, i, j; return)

i: a segment number

j: a segment number

returns: the returned number of end-of-message segments in the range of segments from number i to number j, inclusive if i<=j

0 in all other cases

A data transmit process obtains information on the amount of transmit data available from the session control by executing the following function:

LAST (port id; return)

return: the highest segment number that could be assigned to data available for transmission from Session Control

6.9 Transmit Allocation Module

Each transmit process must call this module to obtain permission to transmit a message. This module guarantees the fair use of Routing resources across all logical links. The algorithms executed by this module are system-dependent and are, therefore, beyond the scope of this specification (Appendix D). The argument "port id" is a port identifier.

A transmit process requests permission to transmit by issuing the following call:

ALLOCATE (port id)

A transmit process checks to see if it has permission to transmit by executing the following Boolean function:

CHECK-ALLOCATE (port id)

returns: true if a message may be sent

false if a message may not be sent

The transmit allocation module cannot give permission to transmit to more than one transmit process at a time.

A transmit process indicates that it no longer needs to transmit by issuing the following call:

DEALLOCATE (port id)

After obtaining permission to transmit, a transmit process must issue this call or the next call before permission to transmit can be given to another transmit process.

A transmit process indicates that it has used its permission to attempt a transmit, but that it also has more data to send by issuing the following call:

REALLOCATE (port id)

7.0 ALGORITHMS

This section contains an overview of some algorithms collectively executed by several NSP components. These algorithms are explicitly defined in Section 6 in the model description. The explanation in this section is added as an aid to understanding.

7.1 Data Segment Retransmission

The data retransmission algorithm described in Section 6.6 in several of the modules operates as follows. There is only one timer for each port. When a Data Segment is transmitted (or retransmitted), start the timer if the segment being sent is the "oldest" segment. That is, the segment number is one greater than the highest segment acknowledged from the remote receiver. Also, restart the timer upon receipt of a data acknowledgment that acknowledges data segments that have not previously been acknowledged but that does not acknowledge all outstanding data segments. Stop the timer upon receipt of acknowledgment of all outstanding segments.

When a data transmit process detects that a timer has expired, that process sets the number of the next Data Segment that can be transmitted to one greater than the value of the highest segment that has been acknowledged from the remote receiver. This causes retransmission if the flow control variables allow retransmission. It will not necessarily cause a retransmission, however, because there may have been a change to the flow control variables.

An implementation of NSP may elect to provide an algorithm different from the one described above. An algorithm that times each outstanding Data Segment separately would provide a higher level of service (in terms of average delay seen by end users) at a cost of more data base storage for NSP.

7.2 Other-Data Handling

Handle other-data subchannel transmission as follows. No more than one other-data message is outstanding at a time for a given port. The variable OTHERstate contains the state of the port with respect to an other-data message. It may have the following states:

State	Meaning
"ready"	No other-data message has been sent but not acknowledged.
"sent"	An other-data message has been sent, has not been acknowledged, and is being timed.
"timeout"	An other-data message has been sent, has not been acknowledged, and has timed out.

When the port is in a state other than "ready," the variable OTHERtyp contains the other-data message type that has been transmitted, and the variable NUMoth contains its number.

Other-data subchannel receiving is handled as follows. The receipt of either a Data Request message or an Interrupt Request message causes an update of the corresponding request count variable in the port (FLOWrem_dat and FLOWrem_int, respectively). The receipt of an Interrupt message causes the placement of interrupt data in BUFrcv.

Since this implementation model can buffer only one received Interrupt message at a time, handle flow control for Interrupt data as follows. There is an interrupt flow control state machine conceptually attached to BUFrcv. This machine has three states. The current state of the machine is contained in variable FLOWloc_int. The states are:

State	Meaning
"empty"	BUFrcv is empty, and an Interrupt Request message has been sent or the logical link has just entered the RUN state (in which there is an implied request for one Interrupt message).
"interrupt"	BUFrcv contains the data from an Interrupt message, and session control has not yet issued an INTERRUPT-RCV call to get the data.
"send request"	BUFrcv is empty, and an Interrupt Request message should be sent to request an additional Interrupt message.

Because of this model for interrupt flow control, an Interrupt Request message cannot be sent for the first time unless FLOWloc_int = "send request" and OTHERstate = "ready."

An implementation of NSP may elect to use a different algorithm for other-data error and flow control from the ones described. An implementation could time each outstanding Other-Data message separately. This would provide a higher level of service (in terms of average delay seen by end users) at a cost of more data base storage for NSP. An implementation could buffer more than one Interrupt message concurrently. The only restriction in the operation of interrupt flow control is that, unlike normal data flow control, an Interrupt Request message cannot be sent unless space is guaranteed for receipt of the interrupt data requested.

7.3 Retransmission Timer Value Estimation

NSP must compute the appropriate value for the time within which to retransmit certain messages. If the value is too great, end users may detect unusually long delays, since Routing may drop packets. If the value is too small, NSP may make very inefficient use of the Routing bandwidth.

NSP attempts to maintain an estimate of the current round trip delay to each remote node with which it is communicating. Variable NODEdelay in a node descriptor contains this value. The estimate is updated each time an observation of round trip delay is made. An observation can be made whenever NSP receives a message in response to a previously transmitted message. Whenever NSP sends a Connect Initiate, Connect Confirm, or Disconnect Initiate message, it saves the current time of day in variable DELAYstr_tim. Whenever NSP receives a Connect Acknowledgment, Connect Confirm, Disconnect Complete message, or any message acknowledging a Connect Confirm, NSP observes the round trip delay to be the current time minus the value in DELAYstr_tim.

In addition, sample round trip delays are observed by timing selected, transmitted Data Segment messages. To conserve space, use only a single timer per port. An implementation may choose to operate with multiple timers. Such operation would tend to produce better estimates at a cost of more data base storage.

Observe the Data Segment timing by starting the timer (provided it is not already running) when a Data Segment message needs to be transmitted the first time and by stopping the timer when an (explicit or implicit) acknowledgment is received for the message. Do not restart the timer if the Data Segment is retransmitted, since the algorithm is attempting to estimate the average delay from first need to transmit to eventual acknowledgment. Once an observed round trip sample is taken as described above, average the value with the current estimate by means of a weighting factor. The formula for this is:

$$\text{NODEdelay} = \frac{\text{NSPweight} * \text{NODEdelay} + \text{deltaT}}{\text{NSPweight} + 1}$$

$$= \text{NODEdelay} + \frac{\text{deltaT} - \text{NODEdelay}}{\text{NSPweight} + 1}$$

where: NSPweight = the weighting factor
NODEdelay = the estimated round trip time
deltaT = the sample round trip time

Note that if NSPweight is equal to a power of 2 minus 1, then this computation can be performed easily.

The time that NSP uses to determine when to retransmit a message is a constant times the estimated round trip delay time. This constant is the "delay factor" and is contained in variable NSPdelay in Table 3. The delay factor and the weighting factor are NSP parameters. NSPweight is an integer in the range 0 to 255, inclusive. NSPdelay is a value in sixteenths of a unit in the range 0 to 15 and 15/16, inclusive.

7.4 Inactivity Timing

If two NSPs cannot communicate with each other for a sufficiently long time (for example, because the network is disconnected), the following problem results. An end user that is either not using a logical link or is passively waiting to receive would not necessarily know that it is uselessly consuming resources by maintaining the logical link. Therefore, NSP contains an algorithm to exercise the logical link when there is no received traffic (either data or NSP control messages) from the remote NSP for each logical link.

The inactivity timing algorithm operates as follows. Start an "inactivity" timer when a logical link enters the RUNNING state. Restart the timer whenever a message is received for the logical link. If the timer expires, NSP attempts to send a Data Request message that does not change the remote NSP's flow control variables. If communications are not possible to the remote NSP, then the retransmission algorithm causes NSP to periodically retransmit the Data Request message. Retransmission can result in a change to the CONFIDENCE variable as described below.

7.5 Confidence Testing

A given NSP module cannot know whether or not a network path exists between it and a given second NSP module, even if the two modules have communicated in the past. Therefore, NSP cannot give a "network disconnection" signal to Session Control when the physical network supporting a logical link fails.

To provide some useful information to Session Control, NSP maintains a counter in variable COUNTretrans in a port. Each time a message timeout occurs (for a Connect Confirm, Disconnect Initiate, Data Segment, Link Service, or Interrupt message) NSP increments this variable and compares it to a global "retransmit threshold" (NSPretrans). If COUNTretrans is greater, then NSP sets variable CONFIDENCE false. Whenever NSP receives an acknowledgment of a previously unacknowledged message, NSP sets CONFIDENCE to true and COUNTretrans to zero. NSP returns the CONFIDENCE variable to Session Control on a CONFIDENCE call.

When communicating with a Version 3.1 neighbor, an implementation of this specification may set CONFIDENCE false immediately upon detection of a failure of the physical link to the neighbor.

8.0 MESSAGE FORMATS

This section specifies the formats for NSP messages.

8.1 Message Format Notation

The following notation is used to describe the messages contained herein:

FIELD (LENGTH) : CODING = description of field

where:

FIELD Is the name of the field being described

LENGTH Is the length of the field as:

1. A number meaning number of 8-bit bytes (octets)

2. A number followed by a "B" meaning number of bits

3. The letters "EX-n" means extensible field. n is a number that specifies the maximum length of 8-bit bytes in the protocol before interpretation, as described below. If no number is specified, the current maximum length is 1 byte. Extensible fields are variable in length consisting of 8-bit bytes. The high-order bit of each byte indicates whether the next byte is part of the same field. A 1 means the next byte is part of this field. A 0 indicates the next byte is the last byte. The low-order 7-bits of each byte are information bits. Extensible fields can be binary or bit map. If they are binary, then 7-bits from each byte are concatenated into a single binary field. If they are bit map, then 7-bits from each byte are used independently or in groups as information bits.

NOTE

The bit definitions define the information bits after removing the extension bits and compressing the bytes.

4. The letters "I-n" means this is an image field. n is a number specifying the maximum length of 8-bit bytes in the image. A 1-byte count of the length of the remainder of the field precedes the image. Image fields are variable in length and may be null (count = 0). All 8-bits of each byte are

information bits. The meaning and interpretation of each image field is defined with that specific field.

CODING Is the representation type used as follows:

A	7-bit ASCII
B	Binary
BM	Bit map (each bit or group of bits has independent meaning)
C	Constant
null	Interpretation data dependent

NOTES

1. If both the length and coding are omitted, the field represents a generic field with a number of subfields specified in the description.

2. Any bit or field specified as "reserved" must be zero unless otherwise noted.

3. All numeric values in this section are decimal unless otherwise noted.

4. Bits are numbered with bit 0 on the right (low-order, least-significant bit) and bit 7 on the left (high-order, most-significant bit). For convenience, when the graphic form of a 2-byte field is given, it will be shown converted to a 16-bit word. When a subfield of a message field contains more than one bit, it should be considered a binary value.

5. Unless otherwise specified, the numbers that appear at the top of the message formats represent bit positions.

6. Bracketed fields are optional.

8.2 General Message Format

In general, NSP messages have the following format:

```
+--------+---------+
| MSGFLG | MSGDATA |
+--------+---------+
```

MSGFLG (EX) : BM   Is a group of fields describing the characteristics of the message. The MSGFLG format is:

```
         +---+---+---+---+---+---+---+---+
Bit:     | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
         +---+---+---+---+---+---+---+---+
Set to:  | 0 |  SUBTYPE  |  TYPE | 0 | 0 |
         +---+---+---+---+---+---+---+---+
```

SUBTYPE (3B) : B Is the message subtype - used to modify TYPE field.

Type	Subtype (Bits)	Bit set to/ Meaning
0	4	0 Data Segment
		1 Interrupt or Link Service
	5	1 Beginning-of-Message segment (bit 4 = 0)
	6	1 End-of-Message segment (bit 4 = 0)
	5	0 Link Service (bit 4 = 1)
	5	1 Interrupt (bit 4 = 1)
	6	0 reserved (bit 4 = 1)
1	4-5	0 Data Acknowledgment
		1 Other-Data Acknowledgment
		2 Connect Acknowledgment
		3 reserved
	6	0 reserved
2	4-6	control type (binary):
		0 No Operation (included for compatibility with NSP 3.1)
		1 Connect Initiate
		2 Connect Confirm
		3 Disconnect Initiate
		4 Disconnect Confirm
		5 reserved (Phase II node init)
		6 Retransmitted Connect Initiate
		7 reserved

TYPE (2B) : B Is the message type (binary)

0	data message
1	acknowledgment message
2	control message
3	reserved

MSGDATA Is the remainder of an NSP message (Section 8. 3-5).

8.3 Data Messages

There are three types of data messages:

1. Data Segment messages (Section 8.3.1)
2. Interrupt messages (Section 8.3.2)
3. Link Service messages (Section 8.3.3)

8.3.1 <u>Data Segment Message</u> - A Data Segment message has the following form:

```
+--------+---------+---------+----------+----------+--------+----+
| MSGFLG | DSTADDR | SRCADDR | [ACKNUM] | [ACKOTH] | SEGNUM |DATA|
+--------+---------+---------+----------+----------+--------+----+
```

MSGFLG (EX) : BM — Represents the message identifier. The format of this field is:

```
         +---+---+---+---+---+---+---+---+
Bit:     | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
         +---+---+---+---+---+---+---+---+
Set to:  | 0 |EOM|BOM| 0 | 0 | 0 | 0 | 0 |
         +---+---+---+---+---+---+---+---+
```

where:

EOM (1B) : BM — Is the end-of-message indicator

0 not-end-of-message
1 end-of-message

BOM (1B) : BM — Is the beginning-of-message indicator

0 not-beginning-of-message
1 beginning-of-message

DSTADDR (2) : B — Is the logical link destination address.

SRCADDR (2) : B — Is the logical link source address.

ACKNUM (2) : BM — Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

```
         +----+-----------+-----------------+
Bit:     | 15 | 14     12 | 11            0 |
         +----+-----------+-----------------+
Set to:  | 1  |  QUAL     |     NUMBER      |
         +----+-----------+-----------------+
```

where:

QUAL (3B) : B — Is an acknowledgment qualifier.

0 ACK
1 NAK
2-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

ACKOTH (2) : BM Is the number of the last NSP Other Data message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

```
         +----+------------+------------------+
Bit:     | 15 | 14      12 | 11             0 |
         +----+------------+------------------+
Set to:  | 1  |  QUAL      |     NUMBER       |
         +----+------------+------------------+
```

where:

QUAL (3B) : B Is an acknowledgment qualifier.

0,1 reserved
2   cross sub-channel ACK
3   cross sub-channel NAK
4-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

SEGNUM (2) : BM Is the number of this Data Segment message. The format for this field is:

```
         +----------+---------------+
Bit:     | 15    12 | 11          0 |
         +----------+---------------+
Set to:  |    0     |    NUMBER     |
         +----------+---------------+
```

DATA Is the data to be sent over a logical link. This field is transparent and may use all 8-bits of each byte. The length of the data field is ascertained from the total length of the Data Segment message and consists of all bytes in the message after the SEGNUM field.

8.3.2 Interrupt Message - The Interrupt message has the following form:

MSGFLG	DSTADDR	SRCADDR	[ACKNUM]	[ACKDAT]	SEGNUM	DATA

MSGFLG (EX) : BM Is the message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	1	1	0	0	0	0

DSTADDR (2) : B Is the logical link destination address.

SRCADDR (2) : B Is the logical link source address.

ACKNUM (2) : BM Is the number of the last NSP Interrupt or Link Service message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

Bit:	15	14 12	11 0
Set to:	1	QUAL	NUMBER

where:

QUAL (3B) : B Is an acknowledgment qualifier.

0	ACK
1	NAK
2-7	reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

ACKDAT (2) : BM Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

Bit:	15	14 12	11 0
Set to:	1	QUAL	NUMBER

where:

QUAL (3B) : B Is an acknowledgment qualifier.

0,1 reserved
2 cross sub-channel ACK
3 cross sub-channel NAK
4-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

SEGNUM (2) : BM Is the number of this Interrupt message. The format for this field is:

```
         +----------+---------------+
Bit:     | 15    12 | 11          0 |
         +----------+---------------+
Set to:  |    0     |    NUMBER     |
         +----------+---------------+
```

DATA Is the interrupt data. This field is transparent and may use all 8-bits of each byte. The length of the data field is ascertained from the total length of the Interrupt message and consists of all bytes in the message after the SEGNUM field. Interrupt data may be no longer than 16 bytes.

8.3.3 Link Service Message - The Link Service message has the following form:

```
+--------+--------+--------+---------+---------+--------+--------+-----+
|MSGFLG  |DSTADDR |SRCADDR |[ACKNUM] |[ACKDAT] |SEGNUM  |LSFLAGS |FCVAL|
+--------+--------+--------+---------+---------+--------+--------+-----+
```

MSGFLG (EX) : BM   Is the message identifier. The format of this field is:

```
         +---+---+---+---+---+---+---+---+
Bit:     | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
         +---+---+---+---+---+---+---+---+
Set to:  | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
         +---+---+---+---+---+---+---+---+
```

DSTADDR (2) : B   Is the logical link destination address.

SRCADDR (2) : B   Is the logical link source address.

ACKNUM (2) : BM   Is the number of the last NSP Interrupt or Link Service message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

```
         +----+--------------+---------------+
Bit:     | 15 | 14        12 | 11          0 |
         +----+--------------+---------------+
Set to:  | 1  |     QUAL     |    NUMBER     |
         +----+--------------+---------------+
```

where:

QUAL (3B) : B   Is an acknowledgment qualifier.

0 ACK
1 NAK
2-7 reserved

NUMBER (12B) : B   Is the number of the message being acknowledged.

ACKDAT (2) : BM   Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

```
         +----+--------------+--------------+
Bit:     | 15 | 14        12 | 11         0 |
         +----+--------------+--------------+
Set to:  | 1  |     QUAL     |    NUMBER    |
         +----+--------------+--------------+
```

where:

QUAL (3B) : B Is an acknowledgment qualifier.

0,1 reserved
2 cross sub-channel ACK
3 cross sub-channel NAK
4-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

SEGNUM (2) : BM Is the number of this Link Service message. The format for this field is:

```
         +---------+--------------+
Bit:     | 15   12 | 11         0 |
         +---------+--------------+
Set to:  |    0    |    NUMBER    |
         +---------+--------------+
```

LSFLAGS (EQ) : BM Is the Link Service flags. The format for this field is as follows:

```
         +---+---+---+---+-----------+--------+
Bit:     | 7 | 6 | 5 | 4 | 3       2 | 1    0 |
         +---+---+---+---+-----------+--------+
Set to:  | 0 | 0 | 0 | 0 | FCVAL INT | FC MOD |
         +---+---+---+---+-----------+--------+
```

where:

FCVAL INT (2B) : B Is the interpretation of FCVAL field

0 data segment or message request count
1 interrupt request count
2-3 reserved

FC MOD (2B) : B Is the flow control modification. If FCVAL INT = 0, then this field has the following contents.

0 no change
1 do not send data
2 send data
3 reserved

If FCVAL INT = 1, then this field is 0 on transmit and ignored on receive.

FCVAL (1) : B Is the number of Session Control messages, Data Segment messages, or Interrupt messages that the sender of the message can receive in addition to those previously requested by a Link Services message. This number is added to the request count which is maintained by NSP, to determine how many Session Control messages, Data Segment messages, or Interrupt messages will be transmitted via a logical link.

NOTES

1. If FCVAL INT = 0, the message is a Data Request message.

2. If FCVAL INT = 1, the message is an Interrupt Request message.

3. The transmit request count for segment flow control may be negative. (Negative values are presented in 2's complement form in the FCVAL field.)

4. If FCVAL is for Session Control message or Interrupt message flow control, the count must be positive. Use 0 if there is to be no change in the count.

8.4 Acknowledgment Types

There are three types of acknowledgment messages:

1. Data Acknowledgment message (Section 8.4.1)

2. Other-Data Acknowledgment messages (Section 8.4.2)

3. Connect Acknowledgment messages (Section 8.4.3)

8.4.1 Data Acknowledgment Message - The Data Acknowledgment message has the following form:

```
+---------+---------+---------+--------+-----------+
| MSGFLG | DSTADDR | SRCADDR | ACKNUM | [ACKOTH] |
+---------+---------+---------+--------+-----------+
```

MSGFLG (EX) : BM   Is the message identifier. The format of this field is:

```
         +---+---+---+---+---+---+---+---+
Bit:     | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
         +---+---+---+---+---+---+---+---+
Set to:  | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
         +---+---+---+---+---+---+---+---+
```

DSTADDR (2) : B   Is the logical link destination address.

SRCADDR (2) : B   Is the logical link source address.

ACKNUM (2) : BM   Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is required. The format for this field is as follows:

```
         +----+-------------+-----------------+
Bit:     | 15 | 14       12 | 11            0 |
         +----+-------------+-----------------+
Set to:  | 1  |    QUAL     |     NUMBER      |
         +----+-------------+-----------------+
```

where:

QUAL (3B) : B   Is an acknowledgment qualifier.

0 ACK
1 NAK
2-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

ACKOTH (2) : BM

Is the number of the last NSP Other Data message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

```
         +----+------------+----------------+
Bit:     | 15 | 14      12 | 11           0 |
         +----+------------+----------------+
Set to:  | 1  |    QUAL    |     NUMBER     |
         +----+------------+----------------+
```

where:

QUAL (3B) : B Is an acknowledgment qualifier.

0,1 reserved
2 cross sub-channel ACK
3 cross sub-channel NAK
4-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

8.4.2 <u>Other-Data Acknowledgment Message</u> - The Other-Data Acknowledgment message acknowledges Interrupt and Link Service messages. It has the following form:

```
+--------+---------+--------+--------+----------+
| MSGFLG | DSTADDR | SRCADDR | ACKNUM | [ACKDAT] |
+--------+---------+--------+--------+----------+
```

MSGFLG (EX) : BM   Is the message identifier. The format of this field is:

```
         +---+---+---+---+---+---+---+---+
Bit:     | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
         +---+---+---+---+---+---+---+---+
Set to:  | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
         +---+---+---+---+---+---+---+---+
```

DSTADDR (2) : B   Is the logical link destination address.

SRCADDR (2) : B   Is the logical link source address.

ACKNUM (2) : BM   Is the number of the last NSP Interrupt or Link Service message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is required. The format for this field is as follows:

```
         +----+----------+--------------+
Bit:     | 15 | 14    12 | 11         0 |
         +----+----------+--------------+
Set to:  | 1  |   QUAL   |    NUMBER    |
         +----+----------+--------------+
```

where:

QUAL (3B) : B   Is an acknowledgment qualifier.

0 ACK
1 NAK
2-7 reserved

NUMBER (12B) : B   Is the number of the message being acknowledged.

ACKDAT (2) : BM   Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

```
         +----+----------+---------------+
Bit:     | 15 | 14    12 | 11          0 |
         +----+----------+---------------+
Set to:  | 1  |   QUAL   |    NUMBER     |
         +----+----------+---------------+
```

where:

QUAL (3B) : B   Is an acknowledgment qualifier.

0,1 reserved
2   cross sub-channel ACK
3   cross sub-channel NAK
4-7 reserved

NUMBER (12B) : B   Is the number of the message being acknowledged.

8.4.3 <u>Connect Acknowledgment Message</u> - The Connect Acknowledgment message has the following form:

```
+--------+--------+
| MSGFLG | DSTADDR|
+--------+--------+
```

MSGFLG (EX) : BM   Is the message identifier. The format of this field is:

```
         +---+---+---+---+---+---+---+---+
Bit:     | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
         +---+---+---+---+---+---+---+---+
Set to:  | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
         +---+---+---+---+---+---+---+---+
```

DSTADDR (2) : B   Is the logical link destination address.

8.5 Control Messages

There are five types of control messages:

1. No Operation messages (Section 8.5.1)
2. Connect Initiate messages (Section 8.5.2)
3. Connect Confirm message (Section 8.5.3)
4. Disconnect Initiate messages (Section 8.5.4)
5. Disconnect Confirm messages (Section 8.5.5)

8.5.1 No Operation Message -

```
+--------+--------+
| MSGFLG | TSTDATA|
+--------+--------+
```

where:

MSGFLG (EX) : BM Is the message identifier. The format of this field is:

```
         +---+---+---+---+---+---+---+---+
Bit:     | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
         +---+---+---+---+---+---+---+---+
Set to:  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
         +---+---+---+---+---+---+---+---+
```

TSTDATA Is any data.

8.5.2 Connect Initiate And Retransmitted Connect Initiate Messages - The Connect Initiate and the Retransmitted Connect Initiate messages have the following form:

MSGFLG	DSTADDR	SRCADDR	SERVICES	INFO	SEGSIZE	DATA-CTL

where:

MSGFLG (EX) : BM Is the Connect Initiate message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	1	1	0	0	0

or

MSGFLG (EX) : BM Is the Retransmitted Connect Initiate message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	1	1	0	1	0	0	0

DSTADDR (2) : B Is the destination logical link address. This address will be 0 to allow the receiving NSP to assign a number dynamically.

SRCADDR (2) : B Is the source logical link address. This number is assigned by the sending NSP and will be used by the destination to address all messages for this logical link. The value 0 is illegal.

SERVICES (EX) : BM The requested services. The format for this field is as follows:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	0	FCOPT		0	1

where:

FCOPT (2B) : B Are the flow control options.

0 none
1 segment request count
2 Session Control message request count
3 reserved

INFO (EX) : BM Is the information. The format for this field is as follows:

```
         +---+---+---+---+---+---+---+---+
Bit:     | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
         +---+---+---+---+---+---+---+---+
Set to:  | 0 | 0 | 0 | 0 | 0 | 0 |  VER  |
         +---+---+---+---+---+---+---+---+
```

where:

VER (2B) : B Is the NSP version.

0 version 3.2
1 version 3.1
2 version 4.0
3 reserved

SEGSIZE (2) : B Is the maximum size (in bytes) of the data in a Data Segment that can be received on this logical link.

DATA-CTL Is the Connect Initiate data field. The length of this field is ascertained from the total length of the Connect Initiate message and consists of all bytes in the message after the SEGSIZE field.

8.5.3 Connect Confirm Message - The Connect Confirm message has the following form:

MSGFLG	DSTADDR	SRCADDR	SERVICES	INFO	SEGSIZE	DATA-CTL

where:

MSGFLG (EX) : BM Is the message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	1	0	1	0	0	0

DSTADDR (2) : B Is the destination logical link address. This will not be 0. It is the value of the SRCADDR field from the Connect Initiate message.

SRCADDR (2) : B Is the source logical link address. This number is assigned by the sending NSP and will be used to address all messages for this logical link. The value 0 is illegal.

SERVICES (EX) : BM Are the requested services. The format for this field is as follows:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	0	FCOPT		0	1

where:

FCOPT (2B) : B Are the flow control options.

0 none
1 segment request count
2 Session Control message request count
3 reserved

INFO (EX) : BM Is the information. The format for this field is as follows:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	0	0	0	VER	

where:

VER (2B) : B Is the NSP version.

0	version 3.2
1	version 3.1
2	version 4.0
3	reserved

SEGSIZE (2) : B Is the maximum size (in bytes) of the data in a Data Segment that can be received on this logical link.

DATA-CTL (I-16) : B Is user-supplied data.

8.5.4 Disconnect Initiate Message - The Disconnect Initiate message has the following form:

```
+-------+---------+---------+--------+---------+
|MSGFLG | DSTADDR | SRCADDR | REASON | DATA-CTL|
+-------+---------+---------+--------+---------+
```

where:

MSGFLG (EX) : BM Is the message identifier. The format of this field is:

```
          +---+---+---+---+---+---+---+---+
Bit:      | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
          +---+---+---+---+---+---+---+---+
Set to:   | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
          +---+---+---+---+---+---+---+---+
```

DSTADDR (2) : B Is the logical link destination address.

SRCADDR (2) : B Is the logical link source address.

REASON (2) : B Is the first two bytes of Session Control disconnect data.

DATA-CTL (I-16) : B Is the remaining bytes of Session Control disconnect data.

8.5.5 Disconnect Confirm Message - A Disconnect Confirm message has the following form:

MSGFLG	DSTADDR	SRCADDR	REASON

where:

MSGFLG (EX) : BM — Is the message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	1	0	0	1	0	0	0

DSTADDR (2) : B — Is the logical link destination address.

SRCADDR (2) : B — Is the logical link source address.

REASON (2) : B — Is the disconnect reason.

NOTES

1. If REASON = 1, the message is a No Resources message.
2. If REASON = 42, the message is a Disconnect Complete message.
3. If REASON = 41, the message is a No Link Terminate message.

APPENDIX A

LQGICAL LINK ADDRESS ASSIGNMENT/DEASSIGNMENT

A logical link address is a 16-bit value. When an NSP module opens a port, it assigns a logical link address. When an NSP module closes a port, it deassigns a logical link address. The algorithm that assigns and deassigns these addresses is implementation-dependent. There are two requirements for this algorithm:

1. It must not assign a given 16-bit address to two ports concurrently, and

2. It must not reassign a given 16-bit address for a long period following its deassignment.

In addition, the algorithm should operate with a modest amount of memory, trading off the amount of memory for the period of reassignment.

The algorithm described in this appendix is a sample algorithm that meets these requirements. No implementation of NSP is required to use this algorithm, however. Any algorithm that meets the two requirements stated above is acceptable. The sample algorithm restricts the number of outstanding, assigned link addresses.

A.1 INTERFACE TO THE ALGORITHM

The sample algorithm is implemented by a module that accepts three calls: one to assign a link address, one to deassign a link address, and one to initialize the module.

The following routine assigns a link address.

GET-ADDRESS

```
        returns:  success - a link address is returned
                  failure - too many link addresses currently assigned
```

The following routine deassigns a link address.

RELEASE-ADDRESS (address)

address: the link address to be deassigned

returns: success
failure - link address was not assigned

The following routine initializes the algorithm module.

INITIALIZE-ADDRESS

NSP calls this routine during NSP initialization. The routine allows the algorithm module to meet the second requirement above even across NSP initializations.

A.2 DATA STRUCTURES

This algorithm forms link addresses of the following form:

random part	index part
r bits	i bits

where:

r+i = 16

No two concurrently assigned link addresses will contain the same value in the low i bits.

Furthermore, the algorithm restricts the number of addresses that can be assigned concurrently to open ports to:

2^i-1

The data base consists of two vectors and three variables. These are the following.

1. Boolean vector INUSE

   This vector contains 2^i bits. There is one bit for each possible value in the index part of a link address.

   A bit is set to "true" if the corresponding index is in use (i.e., is in the lower i bits of an assigned link address). The bit is set to "false" otherwise.

2. Vector RANDOM

   This vector contains 2^i entries, each r bits wide. An element of the vector contains the random part of the last link address assigned with the index part equal to the index of this element in the vector.

3. Variable NUMBER-ASSIGNED

   This variable contains the number of link addresses currently assigned. It has a value in the following range:

   0 <= NUMBER-ASSIGNED <= 2^i-1

   When NUMBER-ASSIGNED = 2^i-1, then no more link addresses may be assigned.

4. Variable INDEX

   This variable contains the index value portion of the last link address that was assigned.

5. Variable TEMP

   This variable is used to temporarily hold the index value portion of a link address that is being deassigned and in module initialization.

A.3 ALGORITHM OPERATION

This algorithm operation is represented in the same high-level language that was used to represent NSP's operation in the body of this document.

GET-ADDRESS:

```
If (NUMBER-ASSIGNED < 2^i-1) then
   NUMBER-ASSIGNED <-- NUMBER-ASSIGNED + 1
   While (INUSE(INDEX) true) do
      INDEX <-- INDEX + 1 (mod 2^i)
   Endwhile
   RANDOM(INDEX) <-- RANDOM (INDEX) + 1 (mod 2^r)
   INUSE(INDEX) <-- true
   random part of link address <-- RANDOM(INDEX)
   index part of link address <-- INDEX
   return success
Else
   return failure
Endif
```

RELEASE-ADDRESS:

```
TEMP <-- index part of the link address
If (INUSE(TEMP) true
      and RANDOM(TEMP) = random part of link address) then
   INUSE(TEMP) <-- false
   NUMBER-ASSIGNED <-- NUMBER-ASSIGNED - 1
   return success
Else
   return failure
   Endif
```

INITIALIZE-ADDRESS:

```
   TEMP <-- 0
   While (TEMP < 2^i) do
      INUSE(TEMP) <-- false
      RANDOM(TEMP) <-- random number (mod 2^r)
      TEMP <-- TEMP + 1
   Endwhile
   INDEX <-- random number (mod 2^i)
   NUMBER-ASSIGNED <-- 0
```

APPENDIX B

SEGMENTATION MODULE EXAMPLE

This appendix models a segmentation module. The model supports the queuing of multiple outstanding transmit requests for each port.

B.1 DATA STRUCTURES

To support this model, each port requires the addition of the following items:

Port Additions

1. A request queue head.

2. A segment queue head.

3. The segment number of the last segment removed from the segment queue (initial value = 0)

4. A Boolean flag to indicate if the next segment placed on the segment queue will be a beginning-of-message segment (initial value = true).

This model also requires a pool of queue control blocks to hold information about queued transmit requests and outstanding segments.

When a transmit request from Session Control is accepted by the segmentation module, a queue control block is added to the request queue for the port. It contains the following information:

Request Queue Control Block

1. Buffer descriptor from the request.
2. Xmtflag from the request.
3. Highest segment number corresponding to the request.
4. Status ("incomplete" or "complete").

When the data from a single transmit request is segmented, each segment is assigned a queue control block that is added to the segment queue for the port. Each segment queue control block contains the following information:

Segment Queue Control Block

1. Buffer descriptor for the segment.
2. End-of-message flag.
3. Beginning-of-message flag.
4. Segment number assigned to the segment.

The queue pointer cells required in these blocks and in the queue head information in the port are not described but are assumed to allow finding the first control block in each queue, the last control block on each queue, and the control block queued after a given control block.

B.2 OPERATION

DATA-XMT

This routine operates as follows.

1. There must be enough queue control blocks available from the queue control block pool to queue one block to the port's request queue and one or more blocks to the port's segment queue. The total number of blocks required is equal to the length of the transmit buffer divided by SIZEseg (for the segment queue) plus one (if there is a remainder from the previous division) plus one (for the request queue). If there aren't enough blocks available from the pool, the DATA-XMT call is returned as "buffer not queued."

2. If the DATA-XMT call is not rejected, add one control block to the request queue. Store the buffer descriptor and xmtflag values from the call in the block. Set the status to "incomplete."

3. Add a control block to the segment queue. The buffer descriptor for the control block contains the address from the DATA-XMT call and a length equal to the minimum of the length from the call and SIZEseg. Set the beginning-of-message flag to true only if the beginning-of-message flag in the port is true. Set the segment number to the segment number of the preceding block on the queue plus one (if there is a preceding block). Otherwise set the segment number to that contained in the port descriptor plus one.

4. Add the remaining control blocks (if any) to the segment queue. The buffer descriptor reflects the segmentation of the transmit buffer into segments. Each segment except, perhaps, the last is as long as the previous segment. Assign each block a segment number equal to that of the preceding block on the queue plus one. Clear the end-of-message and beginning-of-message flags of each block, except, perhaps, the last one. The last block has the end-of-message flag set only if the xmtflag value in the DATA-XMT call indicates

end-of-message.

5. Give the request queue control block queued in step 2 the segment number of the last block on the segment queue.

XMT-POLL

This routine examines the first block on the request queue. If the status is "complete," remove the block from the queue. Return the block to the pool. Give a "transmit complete" return with the buffer descriptor from the block. If the status is not "complete," give a "no transmit complete" return.

GET-SEGMENT

This routine examines the segment queue to find an entry with a matching segment number. The buffer descriptor, end-of-message flag, and beginning-of-message flag are returned.

ACK-SESSION-CONTROL

This routine operates as follows.

1. Examine the first block on the segment queue. If the segment number from the call is less than the segment number (modulo 4096) in the block, go to step 3. Otherwise, go to step 2.

2. Remove the block from the queue and return the block to the pool. Store the segment number from the block in the port. Go to step 1.

3. Examine the request queue. Mark every entry on the queue containing a segment number less than (modulo 4096) or equal to the segment number from the call "complete." Make a return.

NM

This routine identifies the entries on the segment queue from the entry with a segment number equal to the first argument in the call up to the entry equal to the second argument in the call, inclusive. It counts the number of these entries that have the end-of-message flag set and returns this value.

LAST

Return the segment number of the last block on the segment queue, if such a block exists. Otherwise, return the segment number from the port.

APPENDIX C

REASSEMBLY MODULE EXAMPLE

This appendix contains a model of a reassembly module. This model supports the queuing of multiple outstanding receive requests for each port. It does not support the use of either cache or commit buffers.

C.1 DATA STRUCTURES

To support this model, each port requires the addition of the following items:

Port Additions

- o A request queue head
- o A variable (FLOWreass) used to contain changes to the request count for the port (initial value = 0)
- o A variable (FLOWhigh) to contain the highest segment number (modulo 4096) stored in a session control receive buffer (initial value = 0)
- o A Boolean variable (FLOWdiscard) to indicate if received segments are to be discarded (initial value = false)

This model also requires a pool of queue control blocks to hold information about queued receive requests. When a receive request from session control is accepted by the reassembly module, a queue control block is added to the request queue for the port. It contains the following information:

Request Queue Control Block

- o Buffer descriptor from the request
- o Temporary buffer descriptor to handle the reception of multiple segments into the same receive buffer
- o Rcvflag from the request

- Status ("incomplete," "EOM -- no truncation," "EOM -- truncation," "no EOM -- no truncation," "no EOM -- truncation").

C.2 OPERATION

In the following descriptions, the checking for invalid port states is not described since it is assumed to be clear from the body of the specification.

DATA-RCV

This routine operates as follows.

1. If no truncation was specified and the buffer is smaller than NSPbuf, reject the call.

2. If no more queue control blocks are available, reject the call.

3. Otherwise, store the call parameters in a queue control block. Set the temporary buffer descriptor equal to the request buffer descriptor. Add the block to the receive request queue.

4. If rcvflag indicated no truncation, increment FLOWreass by one. Otherwise, compute the smallest integer greater than or equal to the length of the receive buffer divided by NSPbuf. This is how many segments will fit into the buffer. Add the result to FLOWreass.

RCV-POLL

If the state of the port is DISCONNECT-NOTIFICATION, DISCONNECT-COMPLETE, or CLOSE-NOTIFICATION, set the status of all "incomplete" control blocks on the request queue to either "no EOM -- no truncation" (if rcvflag was "no truncation allowed") or "no EOM -- truncation" (if rcvflag was "truncation allowed").

Examine the first block on the receive queue. If it has a value other than "incomplete," remove the block from the queue. Return the block to the control block pool. Return the request buffer descriptor and status value to Session Control.

If the return block has the value "incomplete," give a "no buffer returned" indication to Session Control.

SPECULATE-NUMBER

Return the contents of FLOWreass and clear FLOWreass.

COMMIT-NUMBER

Return the contents of FLOWhigh.

STORE-SEGMENT

The description of this routine uses a colloquial, high-level language. The terms NUMBER and EOM represent the segment number and end-of-message flags, respectively, passed to this routine by the data receive process.

```
If (NUMBER = FLOWhigh + 1) then
   Find the first "incomplete" queued receive request
   If (such a request exists) then
      FLOWhigh <-- FLOWhigh + 1
      If (rcvflag = "no truncation") then
         Put received data in front of buffer
         Set status using EOM
      Else
         If (FLOWdiscard) then
            If (EOM set) then
               FLOWdiscard <-- false
            Else
               FLOWreass <-- FLOWreass + 1
            Endif
         Else
            Put data (that will fit) in buffer (NOTE 1)
            Adjust temporary buffer descriptor to reflect storage
            If (data fit in buffer) then
               If (EOM set) then
                  Calculate space loss (NOTE 2)
                  FLOW reass <-- FLOWreass -- space loss
                  Set status to "EOM -- no truncation"
               Endif
            Else
               Set status to "EOM -- truncation"
               If (EOM not set) then
                  FLOWreass <-- FLOWreass + 1
                  FLOWdiscard <-- true
               Endif
            Endif
         Endif
      Endif
   Endif
Endif
```

NOTES

1. Use the temporary buffer descriptor.

2. The space loss is equal to the number of segments that were requested to fill the buffer, but for which there will be no receive space due to the impending return of the buffer partially filled.

APPENDIX D

TRANSMIT ALLOCATION MODULE EXAMPLE

This appendix contains a model of a transmit allocation module.

D.1 DATA STRUCTURES

This model requires a list structure. Each element in the list contains a port identifier. This list must be large enough to hold one element for each port that NSP can handle.

D.2 PRIMITIVE FUNCTIONS

This model assumes that the functions described below are available.

List Manipulation Functions

1. An element can be added to a list.

2. An element, selected by either index or entry contents, can be removed from the list.

3. The contents of the first list entry can be read.

Random Number Generation

A random number in a selected range can be obtained.

D.3 OPERATION

The following description of the transmit allocation module operation uses a high-level, colloquial language.

ALLOCATE (port id)

Add an element with port id to the list.

CHECK-ALLOCATE (port id)

```
If (port id = contents of first list entry) then
     return success
Else
     return failure
Endif
```

DEALLOCATE (port id)

```
Remove the list entry containing port id
Call REALLOCATE
```

REALLOCATE (port id)

```
If (list not empty) then
     Get random index (NOTE)
     Swap first and indexed entries
Endif
```

NOTE

This function obtains a random number in the range (1, length of list).

APPENDIX E

REVISION HISTORY

This appendix describes the differences from NSP 3.2.1 to NSP 4.0.

1. Phase III and Phase IV NSP do net send NAKs

2. Several fixes to Section 6.4.2 and 6.8.

3. The variable "NUMsent" was added to the Session Control Port to indicate the number of the highest numbered Data Segment message sent by local NSP. Changes were made to the "PROCESS-DATA-ACK" routine and the Data Transmit Process to accomodate the new variable.

4. Change the names of the Network Services Layer and Transport Layer to the End Communications Layer and Routing Layer, respectively.

5. Addition of the "tryhard" parameter to the Routing Layer interface and a procedure for setting the TRYHARD variable in the Session Control Port data base. This is a local change only and does not affect the NSP protocol.

6. Replace the "channel" parameter with "circuit" and "nexthop" parmaters as required by Routing version 2.0.

7. Changes to the connect procedure to allow for the retransmission of Connect Initiate messages, with new message types for retransmitted Connect Initiate message, so retransmitted "connects" will be ignored by old implementations. This is an upward compatible change.

8. Changes to the flow control procedures and message formats to permit cross sub-channel piggybacking of acknowledgments. Cross sub-channel acknowledgments are not sent, except when communicating with an NSP Vs. 4.0, so this is an upward compatible change.

APPENDIX F

GLOSSARY

confidence

An NSP variable (CONFIDENCE) that indicates the probable connectedness of the physical network supporting a logical link.

data flow

The movement of data from a source Session Control to a destination Session Control. NSP transforms data from Session Control transmit buffers to a network form before sending it across a logical link. NSP retransforms the data at the destination from its network form to its receive buffer form. Data flows in both directions (full-duplex) on a logical link.

datagram

A unit of data, including NSP control information, that is passed to the Routing Layer for transmission to a destination system. When Routing adds its route header information, the unit becomes a packet.

Data Link

The DNA layer below the Routing Layer. The modules in the Data Link Layer manage physical channels and maintain data integrity.

delay factor

An NSP parameter (NSPdelay) that is multiplied by the estimated round trip delay time to determine the appropriate value for the time to retransmit certain NSP messages.

delay weight

An NSP parameter (NSPweight) that is used to calculate a new value of the estimated round trip delay. The old round trip delay is weighted by a function of this statistical factor to calculate the new round trip delay. If the delay weight is set high, the retransmit time changes slowly. If the weight is set low, the observed round trip time can change quickly if the observed round trip delays have a wide variance, and thus the retransmit time can change more rapidly. The default value for delay weight is 3.

Disconnect Confirm

The NSP No Resources, Disconnect Complete, and No Link messages. The REASON field in the Disconnect Confirm message (Section 8) indicates which message applies.

error control

The NSP function that insures the delivery of NSP data messages. It consists of an acknowledgment mechanism.

flow control

The NSP function that coordinates the flow of data on a logical link in both directions, from transmit buffers to receive buffers, in order to minimize communications overhead.

inactivity timer

A timer that, upon expiration, causes NSP to attempt to send a Data Request message. NSP starts this timer when a logical link enters the RUN/RUN state. Whenever NSP receives a Data Request message for that logical link, NSP restarts this timer. The purpose of the timer is to provide activity for the logical link so that NSP can determine the probable connectedness of the physical network supporting the link. The value for the timer is an NSP parameter (TIMERinact).

Link Service

The NSP messages that carry flow control information. These messages are the Data Request and the Interrupt Request messages.

logical link

A virtual channel between two Session Control implementations or between two components of one Session Control implementation. NSP's major function is the creation and destruction of logical links.

logical link identification

A unique 32-bit number describing a logical link. This identification consists of the two 16-bit addresses of the ports at each end of the link.

Network Management

The DNA layer directly above the Session Control Layer that enables operator control over and observation of network parameters and variables. Network Management also provides down-line loading, up-line dumping, and testing functions.

node descriptor

A collection of variables and counters pertaining to communications with a particular node. Some of the variables and counters are the estimated round trip delay, traffic usage counters, and error counters.

Other-Data

The NSP Data Request, Interrupt Request, and Interrupt messages. These are all the NSP data messages other than Data Segment. Because all Other-Data messages move in the same data subchannel, it is sometimes useful to group them together.

port

A collection of control variables and parameters for managing logical links. Each logical link has a port at each end. Each NSP at each node has a numer of available ports. When Session Control requests a logical link or requests a port be opened to receive an incoming connect request, NSP allocates a port if sufficient resources are available.

reassembly

The ordering of received data segments by NSP into numbered sequence for placement into Session Control receive buffers.

request count

This term has two different definitions in the document. 1) Variables (FLOWrem.dat and FLOWrem.int) that NSP uses to determine when to send data. 2) Values sent in Link Service messages. The flow control mechanism adds the request counts received in Data Request and Interrupt Request (Link Service) messages (definition 2, above) to the request counts it maintains (definition 1, above) to determine when to send data.

retransmission

The resending of NSP data messages that have not been acknowledged within a certain period of time. This is part of NSP's error control mechanism.

retransmission counter

An NSP variable (COUNTretrans) that contains a count of message timeouts for Connect Confirm, Disconnect Initiate, Data Segment, Link Service, and Interrupt messages. NSP compares this variable with the retransmit threshold to calculate the confidence variable.

retransmit threshold

An NSP variable (NSPretrans) equal to the maximum number of successive times a retransmission occurs with no intervening, received acknowledgment before NSP decides that the physical network supporting a logical link has failed. NSP compares the retransmit threshold with the retransmission counter to determine the value of the confidence variable.

round trip delay

An NSP parameter (NODEdelay) that represents the current estimated time for an acknowledgment to be received for an NSP message. This parameter is calculated by a formula described in Section 4.7.6.

segment

The data carried in a Data Segment message. NSP divides the data from Session Control transmit buffers into numbered segments for transmission by Routing.

segmentation

The division of normal data from Session Control transmit buffers into numbered segments for transmission over logical links.

Session Control

The DNA layer directly above NSP. Session Control defines the system-dependent aspects of logical link communication. Session Control provides functions such as name to address translation, process addressing, and in some systems, access control.

subchannel

A logical communications path within a logical link that handles a defined category of NSP data messages. Because Data Segment messages are handled differently from Other-Data messages, the two types of messages can be thought of as traveling in two different subchannels.

Routing

The DNA layer directly below NSP that provides NSP with routing, congestion control, and packet lifetime control services.

DECnet Digital Network Architecture Phase IV
NSP Functional Specification
AA-X439A-TK

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- □ Assembly language programmer
- □ Higher-level language programmer
- □ Occasional programmer (experienced)
- □ User with little programming experience
- □ Student programmer
- □ Other (please specify)________________

Name________________ Date________________

Organization________________

Street________________

City________________ State________ Zip Code________
or
Country

- - - - —Do Not Tear - Fold Here and Tape -

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE DOCUMENTATION
1925 ANDOVER STREET TW/E07
TEWKSBURY, MASSACHUSETTS 01876

- - - - Do Not Tear - Fold Here and Tape -

Cut Along Dotted Line

Printed in U.S.A.