Local Area Transport (LAT) Specification

Part No. AA-NL26A-TE

A simple, efficient, transparent model for exchanging data between terminals connected to terminal servers and host operating system processes is described. The model is termed Local Area Transport (LAT). LAT is carefully tailored to take advantage of the environment offered by Local Area Networks, such as the Ethernet data link, but maintains much of the simplicity of traditional methods of connecting terminals and hosts.

digital™

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.

June 1989

The following are trademarks of DIGITAL Equipment Corporation:

DEC	LAT	RT
DECmate	MASSBUS	ULTRIX
DECnet	PDP	UNIBUS
DECUS	P/OS	VAX
DECwriter	Professional	VAXcluster
DIBOL	Rainbow	VMS
digital™	RSTS	VT
DNA	RSX	Work Processor

This manual was produced by Networks and Communications Publications

This document was prepared using VAX DOCUMENT, Version 1.1

Contents

4 Circuit and Session Layers

5 Connection Solicitation

A Service Class 1 - Interactive And Application Terminals.

B Compatibility and Implementation

C Algorithm For Assignment/Deassignment Request/Entry Identifiers

Figures

Tables

Preface

SCOPE

This document presents a communication architecture for an Ethernet local area network. The architecture is called Local Area Transport (LAT). LAT is utilized as a low level communication service upon which other higher level services are layered.

LAT is structured as a communication service for terminal servers and host operating systems. The reason for presenting a specific model is to provide a clear example of how the architecture can be implemented. In fact, the architecture is appropriate to applications other than terminal to host communications. In general the term "terminal" depending upon context means not only conventional interactive terminal, but rather a port with equipment connected to it. Therefore, the term "terminal" refers to a conventional terminal, application terminal, printer, and even a computer.

The document assumes that the reader is familiar with the Ethernet, communications concepts, and practical problems accompanying implementations of distributed services.

The document describes a data transport service provided to the host and the terminal server. The level of detail is sufficient to allow the interoperability of hosts and servers. Specific issues involved in building products that utilize LAT as a transport service are addressed in detail by service classes.

Service classes are documented in appendices. Service classes define message formats and algorithms which extend the basic services provided by the LAT architecture. These extensions address problems that are unique to the service class or to the implementation of a product.

ASSOCIATED DOCUMENTATION

- "The Ethernet - A Local Area Network - Data Link Layer and Physical Layer Specifications", DEC-INTEL-XEROX, V2.0, September 30, 1980.
- DNA CSMA/CD Data Link Functional Specification, Version 1.0.1, 25 November 1985, Digital Equipment Corporation, Order No. AA-Y298A-TK.
- DDNA NI Node Product Architecture Specification, Version 2.0.1, 11 November 1988, Digital Equipment Corporation.
- DEC STD 169 (DEC multinational characters set), May 1982.
- DECnet Digital Network Architecture Phase IV, NSP Functional Specification

PURPOSE

The purpose of this document is to specify the LAT architecture in sufficient detail to allow interoperable implementations to be built based on this document. The purpose of the LAT protocol is to bias every design decision in favor of simplicity, while simultaneously preserving the goals of the LAT architecture.

DOCUMENT STRUCTURE

The LAT architecture document consists of the following chapters:

- Chapter 1 (Introduction) - states the assumptions, goals and development history of the LAT protocol, defines the terminology and notations used in the document.
- Chapter 2 (Architecture Overview) - describes main features, functions and the characteristics of the LAT protocol.
- Chapter 3 (Naming and Translation) - presents syntax and semantic of Names, used by LAT architecture, describes Name Translation Process.
- Chapter 4 (Circuit and Session Layers) - presents an architectural model which describes state diagrams, axioms and algorithms, and messages for the virtual circuit and session establishment and control.
- Chapter 5 (Connection Solicitation) - presents and architectural model which describes connection initiation and queuing processes algorithms and messages.

- Appendix A (Service class 1) - presents interactive and application terminals services: describes the local area directory service, presents service access control, describes algorithms and defines messages and extensions of the slots used by the Service Class 1.

- Appendix B (Compatibility and Implementation) - discusses compatibility issues between products implementing LAT 5.0 and LAT 5.1 versions of the Architecture.

CONVENTIONS USED IN THIS MANUAL

All numeric values are specified in decimal.

Character string literals are quoted as in "DELPHI". Occasionally, phrases and terms that are conceptually important to the architecture are quoted, "balanced mode" for instance.

Capitalized names are architecturally defined, an example is SERVER_CIRCUIT_TIMER. Lower case names are function names or events. For example: transmit_unacknowledged_queue is a function name and Send_data is an event.

1
Introduction

Local area networks allow computing resources to be physically distributed throughout a facility, which satisfies the needs of the facility, instead of the needs of the computing resources. Also local area networks dramatically reduced cabling costs, since all of the distributed computing resources connect to a common cable.

An Ethernet can have as many as 1000 attachments on a single coaxial cable over 1 mile in length. A potential problem with so many attachments is the limited bandwidth available on the Ethernet (about 7 usable megabits/second). For this reason, communication architectures operating in this shared environment should allocate the available bandwidth efficiently, fairly and predictably among the many systems. This is an explicit goal of the LAT architecture.

A non-goal of the LAT architecture is to specify a transport mechanism sufficient for the needs of a large number of applications. Instead, the architecture makes simplifying assumptions appropriate to a subset of possible applications. The most important assumptions are:

- Communication is local to a single (logical)Ethernet. This eliminates the need for any routing capability.
- The nature of the communication is inherently asymmetric. This simplifies connection management, increases efficiency and greatly simplifies the host implementation.
- The bandwidth of the Ethernet is much greater than the bandwidth needed by an application. This assumption results in a timer based protocol.

The above assumptions applied to the problem of connecting terminals to hosts allow the following tradeoffs:

- Minimize the load on the host operating systems by transferring load to the terminal server.
- Reduce terminal server complexity to allow very low cost hardware implementations, or increase the complexity to achieve a value added service in the terminal server.
- Allow a user terminal to attach to any host in the local area, or restrict the users view to a subset of the available hosts.
- Increase the level of performance at the terminal servers and limit the total number of terminal servers simultaneously using the Ethernet, or decrease the level of performance at the terminal servers allowing a greater number of terminal servers to utilize the shared Ethernet.

LAT views the Ethernet as a local device, not as a network. This approach allows the implementation of the architecture to be confined to low levels of the host operating systems. It also minimizes the cost of installation and support of the computing resources by requiring very little training on the part of the network manager and users.

LAT assumes the Ethernet has very predictable attributes. LAT's performance depends on "low probability" events occurring infrequently. LAT's correctness depends on very "low probability events" not occurring at all. If "low probability" means less than one event every hour, and "very low probability" means less than one event every year, then LAT assumes the Ethernet data link has the following attributes:

- a low probability of datagram duplication
- a low probability of datagrams being received in an order different from that in which they were transmitted
- a low probability of datagrams being corrupted (and therefore not delivered)
- a low probability of datagrams being delayed more than 10 milliseconds between source and destination ports
- a very low probability of datagrams being delayed more than 10 seconds between source and destination ports
- a very low probability of datagrams being delivered to the wrong destination address

- a very low probability of datagrams being delivered which contain undetected corrupted data
- a bandwidth greater than 1 megabit/second
- a broadcast/multicast capability (see above)

Another non-goal of this architecture is to provide any level of security beyond that provided by the Ethernet itself. Extensions to this architecture in the areas of authentication and data link encryption have been anticipated, but not realized.

LAT architecture development history is presented by two protocol versions: LAT 5.0 version and the LAT 5.1 version. The main characteristics provided by the LAT 5.0 version (first version implemented in the actual products) are:

- multiplexing multiple sessions over one virtual circuit;
- asymmetry - master/slave relations between terminal server (master) and host (slave) where connection can be established only from master to slave;
- services are offered only by hosts (hosts advertise offered services through multicast messages and never listen to multicasts; servers listen to multicasts and support data base of nodes and services).

The additional features of LAT architecture 5.1 (compared to LAT architecture 5.0) consist of the following:

- support of application terminals (such as printers) that require connections driven by the the hosts (slave) nodes. I.e. LAT 5.1 version allows host (slave) nodes initiate connections to the server (master) nodes.
- in addition to the host advertising, servers can advertise offered services (such as printers) by listening to the solicit information request issued by host nodes and responding with the message containing required information;
- connections to the specific ports to allow users to set characteristics and connect to required ports;
- clarification of the Group Codes, ambiguously defined by the LAT 5.0 architecture.
- compatibility between products implementing 5.0 and 5.1 versions of the architecture.

LAT 5.1 architecture was developed to satisfy the above mentioned requirements. The main features of the LAT 5.1 version discussed in this document are:

- preserving "asymmetrical" nature of the master-slave relations, the LAT 5.1 architecture allows hosts (slaves) to initiate connection to the terminal servers (masters), providing connections to the application terminals;
- symmetry of services was introduced (i.e. servers can advertise offered services as well as hosts). Special measures were included in the architecture to allow hosts to choose means of processing service information depending upon available resources;
- "queued" access to the services allowing services queue connection request to the currently active service for the future processing;
- port names were introduced, allowing a user to directly connect to the specified port offering the requested service;
- usage clarification of the Group Codes as "connectivity restriction" mechanism;
- clarification of the issues concerning session characteristics, data transparency and port characteristics setting;
- discussion of the compatibility issues between products implementing LAT 5.0 and LAT 5.1 architecture.

The major goal of the new LAT architecture is to provide a reasonable compromise between the number and complexity of the features included in the architecture and requirements of the wide range of products based on the LAT architecture. Another very important issue in the process of the development of the LAT 5.1 architecture was to provide smooth transition from the LAT 5.0 products to the LAT 5.1 products. The main issue to resolve was to design the architecture which addresses needs of different products, allows products to satisfy their time-to-market requirements and preserves compatibility between products.

In pursuing these goals major attention in the design of the LAT architecture was paid to two aspects: a)to provide wide spectrum of the LAT products with the architecture to satisfy their needs without necessity to invent new mechanisms or protocols for each product and b)produce a "modular" architecture that would allow any specific product to implement any of new features without necessity to implement all of them and still preserve compatibility across the product space.

The above requirements dictated the architecture would have:

- features that can be "added" to the LAT products without need to rewrite the whole implementation completely (new state tables, changes in the protocol, etc.);

- features that are "modular" (i.e. each product can decide what features should be implemented based on their schedule, resources etc.). It is not necessary to implement all features in order to be compatible with other products;

- compatibility between products based on the LAT 5.0 version of the architecture and new products implementing features of the LAT 5.1 version.

1.1 Terminology

- local area (network) - the topology defined by the set of logically equivalent processors directly attached to a shared interconnect.

- terminal server - a dedicated function system (processor, controller) providing attachment points for terminals in the local area via a responsive virtual circuit service spanning the shared interconnect.

- datagram - an atomic unit of information exchanged by local area networks. In the Ethernet implementation, datagrams are required to have a constant format consisting of: destination port address, source port address, protocol type, data and an error detection code. Datagrams may get corrupted on the Ethernet, and are therefore not always delivered to the destination address.

- message - a datagram under virtual circuit error control.

- slot - a segment of a message used to communicate data between a terminal on a terminal server and a host service. Messages may have zero or more slots.

- session (connection) - a transient association which allows a terminal server to exchange data reliably with a single host service utilizing an underlying shared virtual circuit.

- flow control - a set of rules applied to processes which prevents a transmitting process from sending data to a receiving process that is not prepared to buffer the transmitted data.

- broadcast - as applied to data links, broadcast capability refers to the ability of any one port to address all other ports simultaneously with a single datagram.

- multicast - as applied to data links, multicast capability refers to the ability of any one port to address a sub-set of all other ports simultaneously with a single datagram.

- users - the consumers of the services provided by this architecture. As applied in this document, the term "user" is an abstraction that refers to the set of routines interfacing to the highest level of the architecture. The services provided to users are connection management and data transfer.

- Service class - a 1-byte value in the range 0-255:

  — value 0 - reserved

  — value 1 - reserved for interactive and application terminals (serial byte stream processing).

  — values in the range 2 to 127 reserved for DEC use

  — values in the range 128 to 255 reserved for customers

- Name - a string of ASCII characters meaningful in the context of a client using it. Names are used to provide identification of entities within the LAT architecture that can and need to be identified. Characters within an ASCII string representing the name are constrained as described in the section of the LAT architecture entitled "Specification of Names."

- Resource - an entity or set of entities known to perform a certain set of functions that can be identified, named, and accessed within LAT.

- Master - an addressable process that provides communication attachment points for virtual circuits. The master initiates and controls activity over virtual circuits. The state-table of a master process is defined in the LAT architecture document.

- Slave - an addressable process that provides the passive side of the communication attachment point for a virtual circuit. The slave responds to the master's request. The state-table of a slave process is defined in the LAT architecture document.

- Virtual circuit - a communication path between a master and a slave. A virtual circuit is a bidirectional, sequential, timely, and error-free logical stream of data. On Ethernet, a virtual circuit service is a value added service since the Ethernet data link provides a datagram service.

- Subject - a consumer of the resources, an active initiator of a connection. A subject can initiate and support relations only with objects, not with other subjects. A subject can be a master as well as a slave.

- Object - a provider of the resources, a passive responder to requests for establishing connections. An object does not initiate connections. An object can be a master as well as a slave.

- Session - a transient association that allows a subject to exchange data reliably with an object by utilizing an underlying shared virtual circuit.

- Service - the descriptive name of the resources; the name is used by users to identify a resource and is used by LAT to establish an access path to the resource.

- Node - the environment on the end of the virtual circuit that provides functioning of a master and slave processes. A node can operate as slave, master or both simultaneously. In this document the expression "slave (master) node" really means "a node operating in slave (master) mode." Each node is uniquely identified by name.

- Advertising - the process that allows users to identify names and characteristics of the resources to be used. As applied in this document, the term "advertising process" refers to a certain message-exchange mechanism provided by the LAT architecture. Each node, whether it is a slave, a master or both, can advertise services.

- Interactive terminal - a device that is under the control of the terminal user connected to a node running in master mode.

- Application terminal - a device that is under control of the application process running within a slave environment (in some cases an application device may not have a keyboard or even be a 'terminal' at all - it may be a line printer, a video monitor, a display window etc.).

- Port - an access point that a node present to users. Each port serves as a communication path between a user and a resource. Ports can be named.

2

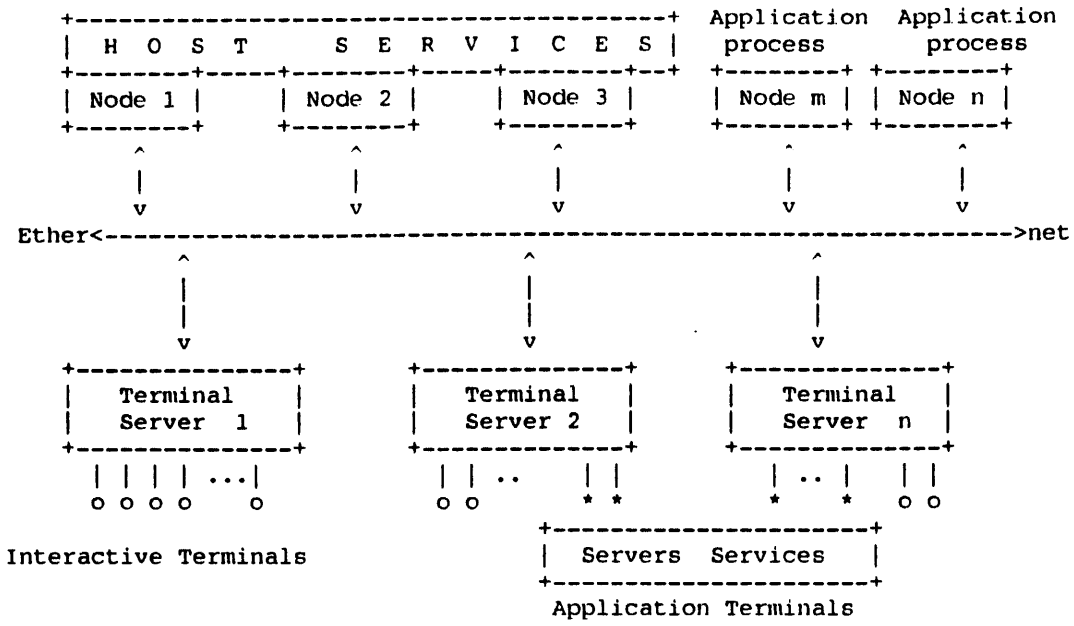
Architecture Overview

2.1 Introduction

The major components of the architecture are the terminal server, the host node, the local area network and software modules in the terminal server and host.

The principal functional capability provided by the architecture is the logical connection of interactive and application terminals to host nodes. A vertical view of the topology would reveal:

Figure 2-1: LAT Network Topology



LAT architecture provides symmetry of services offered by hosts as well as terminal servers. Users on the terminal servers have an access to the list of services available on the hosts and can initiate the connections to the services. The same is true for users (i.e. application processes) on the hosts - they have an access to the list of services offered by the terminal servers and can initiate connections to those services (though because of asymmetry in the algorithms on the host and terminal servers actual connection initiation processes are different as shown further).

Each of the users of the terminals on "terminal server 1" could be connected to a different host service simultaneously. Or they could all be utilizing the same host service. The same holds true for all of the other terminal servers.

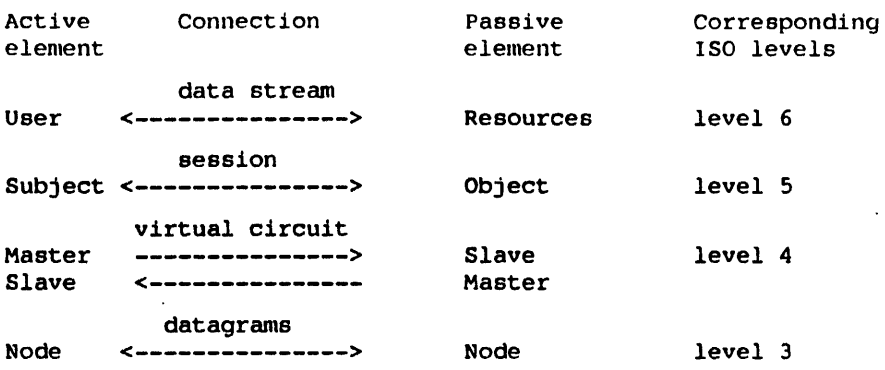
Symmetrically, each application process running on the host node (i.e. "user") can be connected to a different service offered by a terminal servers or several users can utilize the same service.

Services are named resources within the LAT environment. Names are the primary mechanism for users to identify required resources and provide LAT with enough information to arrange an access path from the user to the resources. The LAT architecture presents a "name translation process" that takes place when a name is presented to the LAT session interface This process provides translation of the descriptive name presented by a user into a physical path to a resource.

It is also possible for a single system to operate simultaneously as both a host implementation and a terminal server implementation. If such a system wishes to allow local terminal users to transparently connect to the local host services, Ethernet messages transmitted by the local system must logically be delivered to the local system.

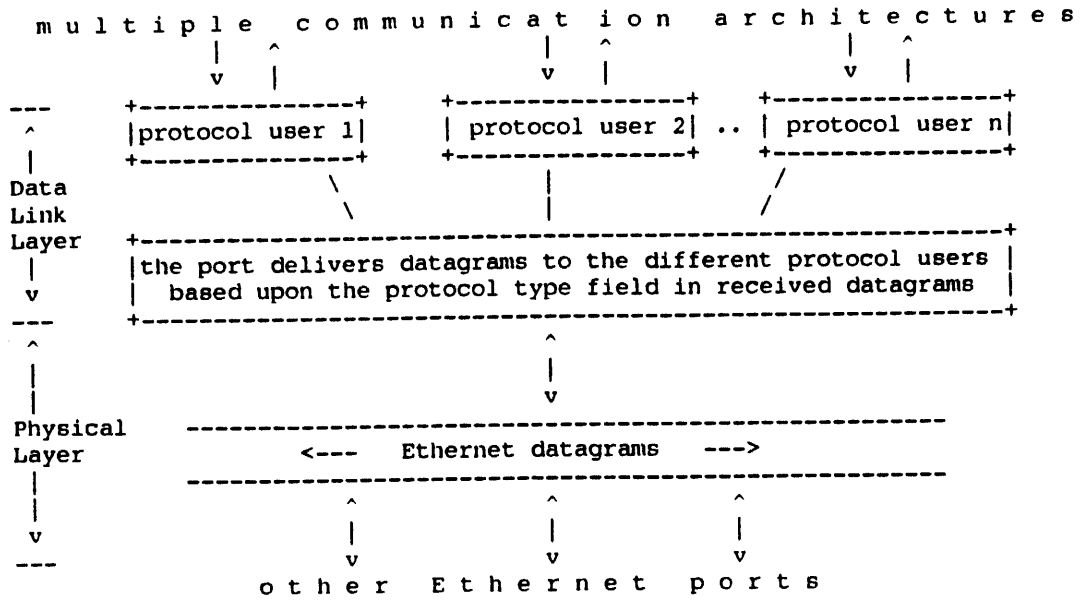
The following diagram shows the relations among master/slave, subject/object, user/resources, and nodes.

Figure 2–2: Relations Between Layers



The Ethernet itself is layered:

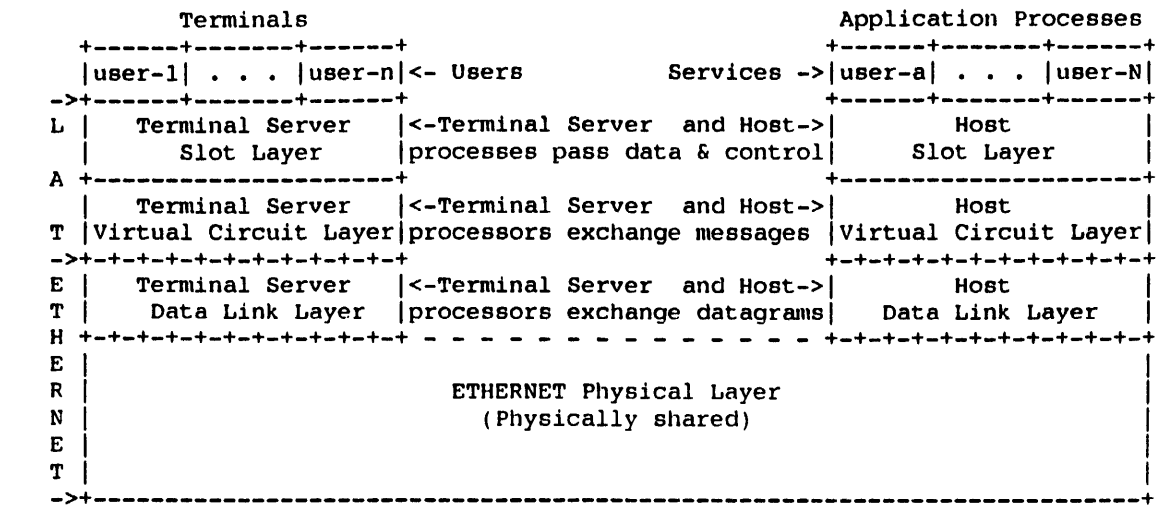
Figure 2–3: Physical and Data Link Layers



As shown above, the Ethernet data link layer (the port hardware and driver) and the Ethernet physical layer (the cable) can simultaneously support LAT and other communication architectures. This is accomplished by assigning each communication architecture a unique protocol type in the Ethernet datagrams. The Ethernet ports use the Ethernet protocol type field of receive datagrams to distinguish between the LAT protocol and other protocol users of the Ethernet.

Datagrams are transmitted and received over the Ethernet by an implementation of the LAT architecture. The LAT architecture could be viewed as a layered architecture:

Figure 2–4: Layered View of the LAT Architecture



In practice, an implementation would collapse the Slot and Virtual Circuit Layers into a single module.

Functionally, the virtual circuit layer establishes and maintains a shared virtual circuit. The slot layer multiplexes one or more users connections over the underlying shared virtual circuit.

2.2 SLOT LAYER - USER INTERFACE

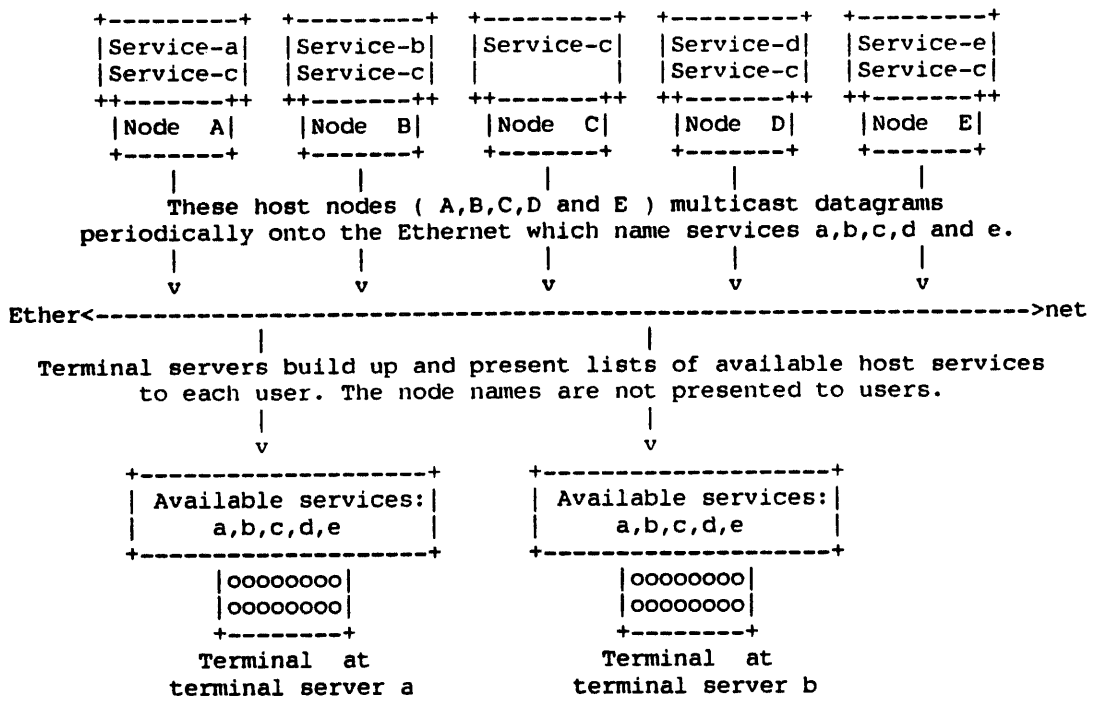
The primary responsibility of the slot layer is user session establishment, data transfer and multiplexing/demultiplexing over a common underlying virtual circuit maintained by the virtual circuit layer. The sessions are established between terminals and host services.

The LAT protocol is specified in a way that allows each service class the freedom to define extensions to the basic connection management and data transfer services. These extensions to the foundation service can address problems unique to the type of service being provided.

2.2.1 Connecting to the Host Service

Each host service makes its presence known to the local area by advertising the service in a datagram which is periodically multicasted to all terminal servers from each host node. Both the host node name (SLAVE_NODE_NAME) and the host service names (SERVICE_NAME) are represented in each multicasted datagram. The terminal servers receive these multicast datagrams, to build up a list of available host nodes and services, so as to provide the user a selection of the hosts services. Normally, these services would be presented at each terminal in a console mode local to the terminal server as shown in Figure 2-5.

Figure 2–5: Connecting to Host Services



A user can simply select the desired host service from the displayed list of available host services. The slot layer translates the selected host service name into the name of a host node which offers the service. After successfully establishing a session with the desired host service, further input typed by the user at the terminal is transferred as if the terminal was, in fact, local to the host. On the other

hand, the peculiarities (such as meaning of control-O, control-T, control-Y ...) of a particular host service remain unknown to the terminal server.

Note that more than one host node can offer the same host service (e.g. host service c). This is useful when the host services being offered are equivalent (e.g. an interactive timesharing service offered by a VAXcluster).

For network management purposes, each terminal server can present a different set of available host services based on group codes assigned to host nodes and terminal servers. This capability is provided to allow segmentation of the computing resources based on such criteria as departmental ownership or physical location. See the section "LOCAL AREA DIRECTORY SERVICE" (Appendix A) for more details.

2.2.2 Connecting to the Terminal Server Service

The case above presented a model for initiating a connection from the interactive terminals (which are connected to terminal servers) to the services offered and advertised by the host operating systems.

LAT architecture also allows the connection of application terminals to terminal servers that offer services. Application terminals are defined as devices under the control of an application process on a host operating system. A line printer connected to a terminal server is an example of an application terminal. An application process such as a line printer despooler can initiate the connection from the host system to the terminal server for this device. Application terminals are not constrained to be output-only devices.

Advertising services, host services periodically multicast advertising messages to the local area network, and terminal servers receive these datagrams to build up a directory of available host nodes and services. A host node would not normally listen to these messages unless it choose to implement both the terminal server and LAT host functionality. In order to not burden host applications with implementing a directory service, LAT architecture provides also a different mechanism that allows host applications to solicit node and service information. Host applications use this mechanism to get information about services and terminal server nodes supporting application devices.

In order to connect an application process running on the host to the service offered by terminal server, the LAT architecture provides two functions:

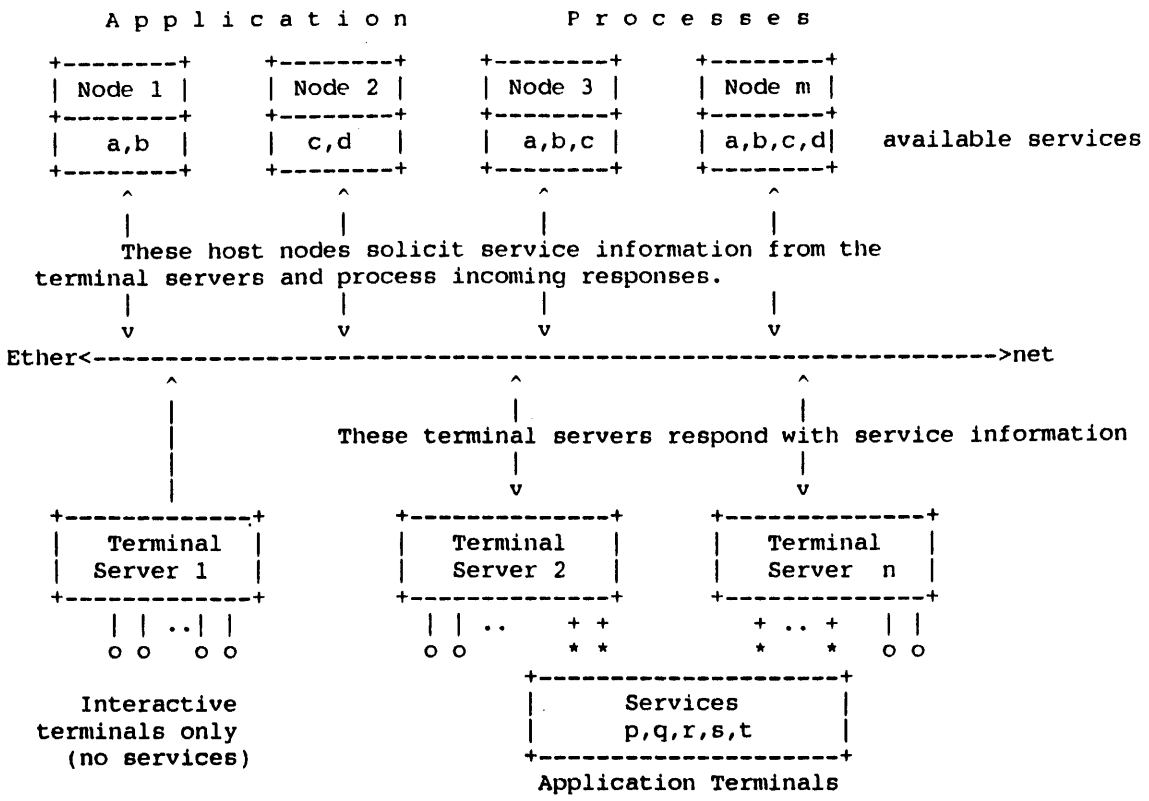
- "connection solicitation". A mechanism that allows host node to solicit connection requests from the terminal server which then actually starts the connection.

- directory service. A mechanism that allows advertising of services offered by servers. That is, the LAT servers can offer services (for example, application terminals) as well as host systems. Advertising services offered by servers allows hosts to receive and process directory information from servers thereby avoiding manual input of the addresses.

An application on any of the host operating systems may solicit a connection to an application terminal by either selecting a service offering the desired application terminal or by selecting a port on a specific terminal server. A service name provides an access path to one or more application terminals or one or more servers. An application terminal may belong to zero or more services. Normally, an application process selects the desired service, and the slot layer translates the service name into the name of a terminal server node which offers that service. The solicit connection request is sent to the target terminal server for evaluation, and any port offering the requested service will satisfy the solicitation connection. Port names are not constrained to be unique within the local area network and are only defined within the context of the port's node. The target terminal server must validate the port name portion of a solicit connection request.

Before a host node solicits a connection to a service, it may multicast a solicit service information request. One or more nodes may respond to the request with service information. The host node evaluates the responses and selects a node to solicit for a connection. If a request is accepted by the terminal server, the terminal server initiates a connection.

Figure 2-6: Connecting to Servers Services



The slot layer of the LAT architecture allows "shared" services, (i.e. one service can be shared by many users by means of queued access to services). This is particularly desirable for services that can only be satisfied by a limited number of physical entities such as those services that offer access to application devices. The principle of "service sharing" is based on a mechanism of queues. Each service possesses "queued" or "non-queued" characteristics. Queues are accessed by subjects through connection requests qualified by "queued" or "non-queued" access methods. The LAT architecture defines methods for a user to request queued or non-queued access to a particular service.

2.3 VIRTUAL CIRCUIT LAYER

The primary responsibility of the virtual circuit layer is to establish and maintain virtual circuits between host nodes and terminal servers.

The virtual circuit allows messages to be reliably exchanged between the terminal server and the host node. The format and the rules governing message exchange is specified by the Local Area Transport architecture. The virtual circuit layer is responsible for translating node names into 48-bit Ethernet addresses. The data

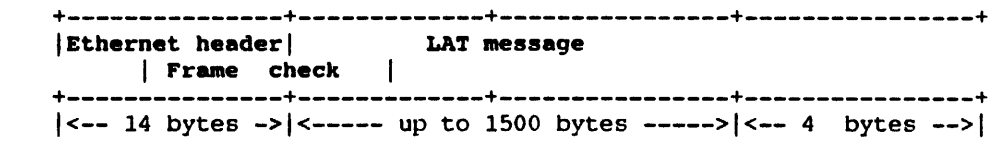
necessary to make this translation is normally supplied to the virtual circuit layer by multicast datagrams.

Transmission of messages from the terminal server to the host node is timer based. The host always responds to these messages from the terminal server. Under certain conditions (see state diagram) the host node may send an unsolicited message.

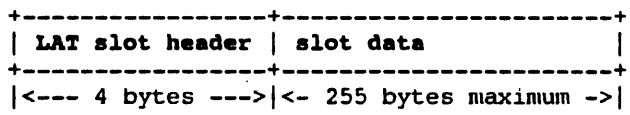
The architecture minimizes the overhead of the virtual circuit by:

- using simple, asymmetric connection management. Only one virtual circuit is established between any pairing of a terminal server and host node (multiple nodes may be implemented in a single processor or multiprocessor). The virtual circuit service is always initiated at the request of the terminal server.
- procrastinating so there is more to do when things have to be done. Messages are not normally exchanged when data is available to be transmitted. Instead, messages are exchanged periodically. The rate of exchange can be set to a constant value or varied to suit the needs of the application. A typical value for terminal servers is about 80 milliseconds. As more users are connected over an existing virtual circuit, the number of messages exchanged is held constant, but the length of each message is likely to increase.
- piggybacking virtual circuit control information and multiple users data in a single message. The virtual circuit simultaneously supports more than one user, the messages are divided into an Ethernet header (which allows the physical cable to be shared), a LAT header (which allows the virtual circuit to be shared) and one or more slots. Slots contain a header, which identifies a terminal and a host process offering the host service.

  An Ethernet message is limited in length to 1518 bytes:



  Maximum size of the LAT message is 1500 bytes. LAT message consists of slots, where a LAT slot is limited to 255 bytes of data:



- assuming a low loss, highly responsive, high bandwidth, point to point interconnect. Messages are not pipelined; instead, each end of a virtual circuit takes turns transmitting messages. This limits the load a single virtual circuit can present to the Ethernet. Since messages can be exchanged quickly, it does not reduce the available bandwidth below useful levels for most applications.

2.4 Product Considerations

2.4.1 Host

Hosts implement the passive (or slave) end of the virtual circuit because this end of the virtual circuit is simpler (and therefore offers less load). The entire architecture can be implemented in operating systems by presenting the LAT user interface to the operating system terminal driver as though it were one or more terminals. By encapsulating the architecture within this existing framework, the maximum benefit can be gained with a minimum effort.

An example sequence of events when a user on the terminal server connects to the service on the host node can be presented follows:

1. A terminal server "hears from" a host node and adds the host node to its menu of available systems and the host service names to its menu of available services.
2. The terminal server user requests a connection to a specific host service by choosing one of the host service names displayed at the terminal server.
3. The terminal server connects to the host node.
4. The operating system terminal driver creates a "virtual" terminal.
5. The terminal server user "logs in".

An example sequence of events when an application process (user) on the host node connects to the service on the terminal server node can be presented follows:

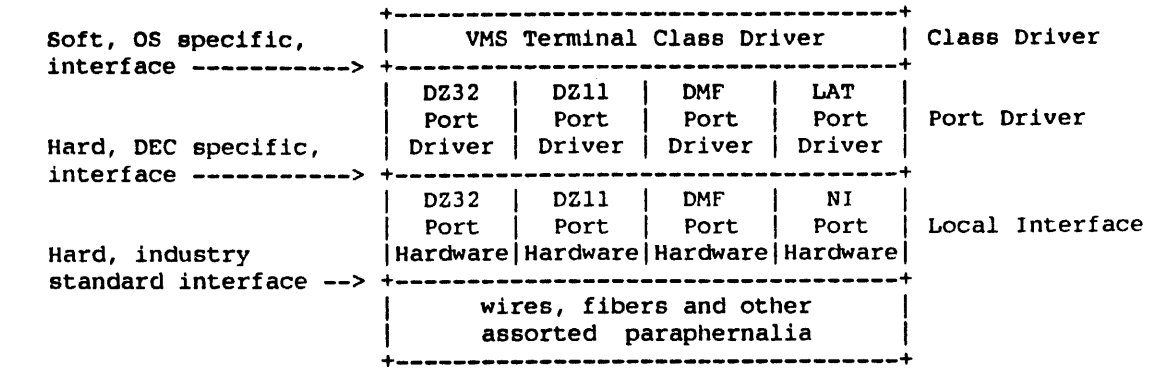
1. The host sends a request, to solicit information about a desirable service from all nodes. The host waits for the responses, processes the information and chooses one particular node.
2. A host sends a connection solicitation request to that particular node and processes a response containing information about status of the request.

3. The terminal server connects to the host node by creating a virtual circuit.

4. The application process starts transmitting data.

Operating systems that implement the LAT architecture will benefit greatly if the terminal driver is organized as a terminal "class driver" and "port driver". The class driver embodies the control characteristics of the operating system terminal interface, while one or more port drivers specialize in passing stream data between the (local) terminal hardware interface and a common class driver interface. As a specific example, consider the following example (VAX/VMS operating system).

Figure 2–7: LAT Driver Organization



Within the host, the implementation of the Local Area Terminal architecture is confined to the box labeled "LAT Port Driver". In an actual implementation, the "LAT Port Driver" would consist of a LAT protocol driver and an underlying, normally shared, NI Port Driver.

2.4.2 Terminal Server

The simplest terminal server can be implemented by using a inexpensive microcomputer. The LAT software modules, and the rest of the operating system code can be implemented in read only memories. A user friendly console interface, better than those found on large commercial terminal switches, can be used to assist the person trying to utilize a host service.

A more complex terminal server could utilize LAT to communicate more structured data than the character streams described in this document. Examples might be workstations transferring files between the host and the workstation, or a terminal server that supported multiple windows at the terminal, each mapped to a different session.

3

Naming and Translation

3.1 Naming Conventions

Service names are the primary mechanism for users to identify required resources and provide LAT with enough information to arrange an access path from the user to the resources. When a service name is presented to the LAT environment, processing takes place in the context of a request that allows the subject to establish a connection with an object and associate the request with an access point (port). Service name translation involves message exchange between subject and object nodes and translation of service names (coded within LAT message fields as ASCII strings) on both communicating nodes as described below.

The scope of the service name space is the local Ethernet or the extended local Ethernet if bridges or repeaters are used to extend the Ethernet local area network.

3.1.1 Service Name Translation Process

A user initiates a connection by presenting a resource name to the name translation process used by LAT. A resource name is based on a triplet of names: the name of a destination node, the name of a requested service, and the name of a port on a destination node. The general principles of the name translation process are presented below, where a connection from the source node (SOURCE_NODE) to the destination node (DESTINATION_NODE) is established based on a triplet of names: SERVICE_NAME, DESTINATION_NODE_NAME, DESTINATION_PORT_NAME. A model of the name translation process is presented in Figure 3-1 ({} denote optional parameters).

Figure 3-1: Name Translation Process

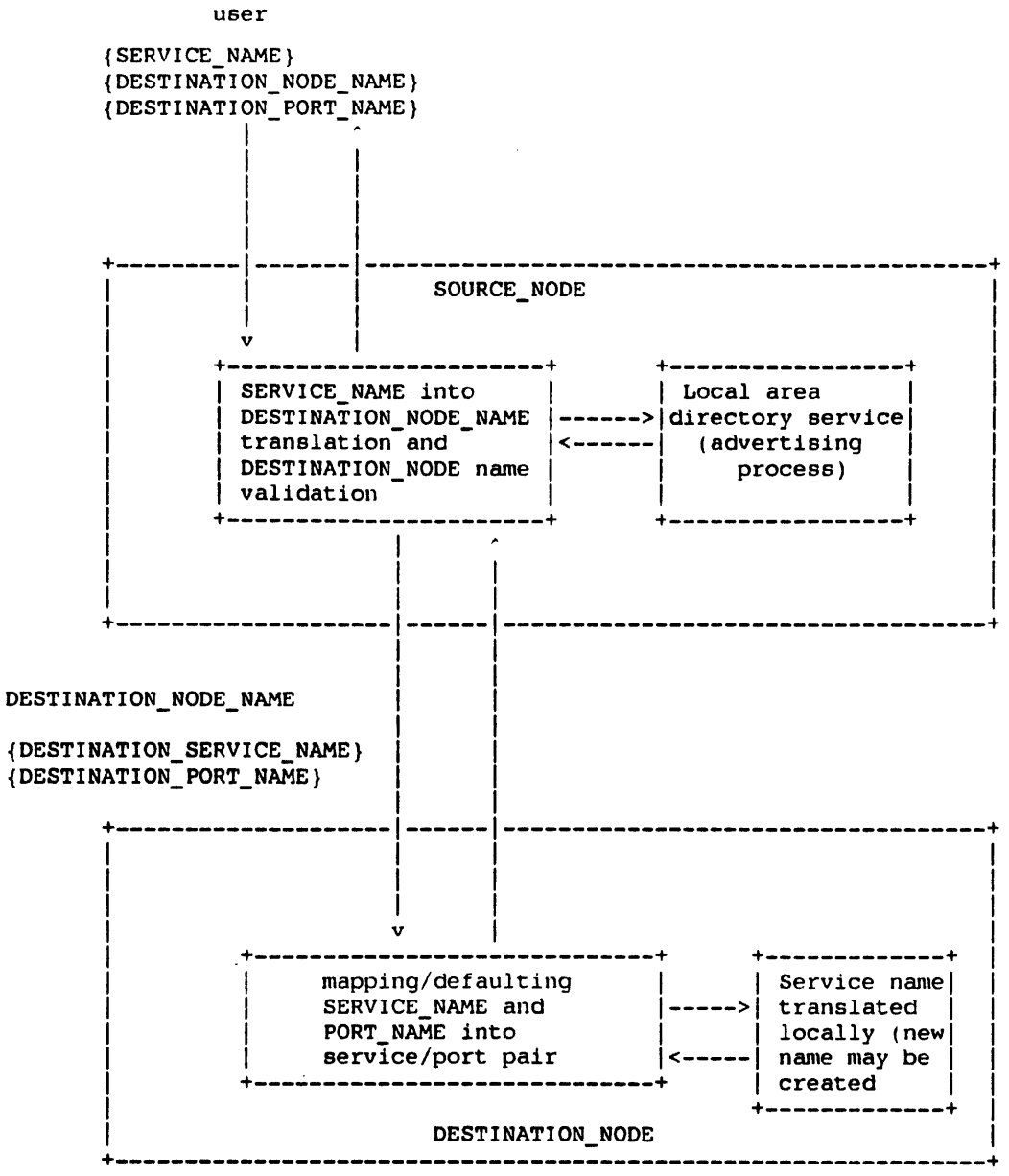
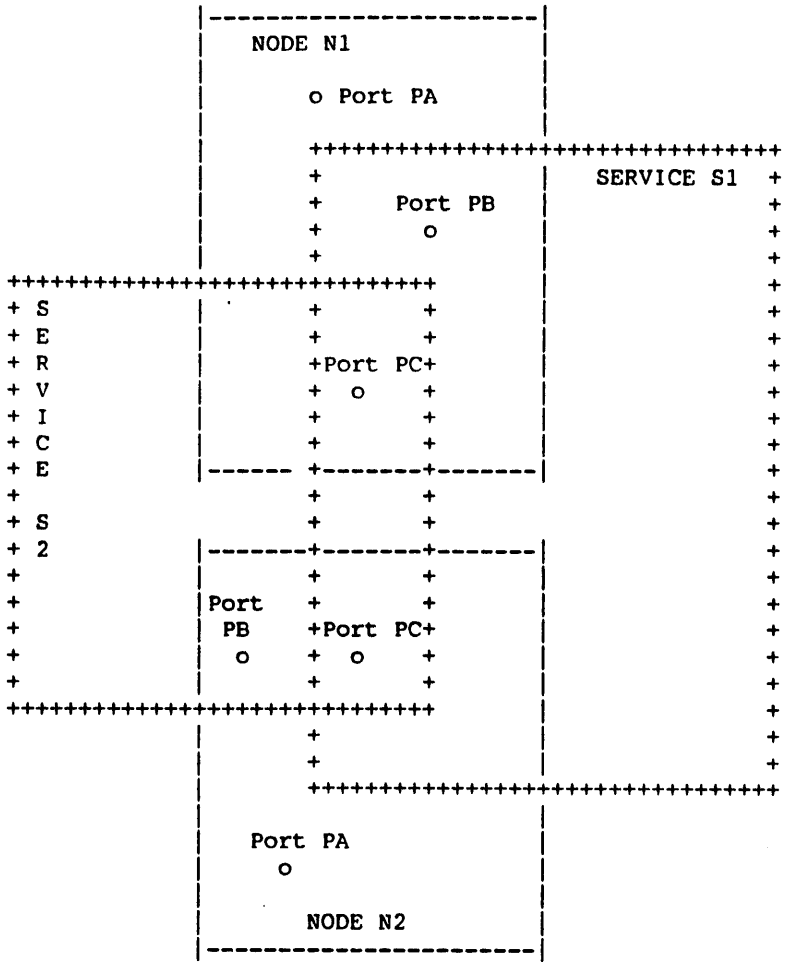


Figure 3-2 presents a possible combination of services, nodes and ports. Table 3-1 gives some examples of the rules of a SERVICE_NAME, NODE_NAME and PORT_NAME translation based on the Figure 3-2.

Services S1 and S2 (denoted by + + + + + +) are offered by Nodes N1 and N2 (denoted by ———). The ports are represented by the symbol "o". There are two services, two nodes, and three ports in each node. Ports PA on Nodes N1 and

N2 do not provide any services. Port PB on node N1 provides only service S1 and Port PB on node N2 provides only service S2. Ports PC on both nodes can provide both services. Each node offers both services S1 and S2. Each node provides a "default service." A default service is a service provided by a node if a received connect request does not specify a destination service name.

Figure 3–2: A Combination of Services, Nodes and Ports



The process that translates the service name on the target node may either cause no translation at all by using the default service name, or causes a complicated translation process depending upon available resources, existing services, schedule, some special translating functions, etc. When it receives the destination service name, the destination node translates it and sends it back to the source node. The source node should not reject the connection if a different service name was returned by a destination node. The source node may use the result of a translation to establish a connection.

Some examples of the translation process are presented in Figure 3-3. Those examples are based on Figure 3-2. Assume that the node translates the service name S1 into S1 and that the default service name is also S1.

Figure 3–3: Name Translation Table (Source Node)

input			result of translation			
service	node	port	service	node	port	error
S1			S1	N1 or N2	PC(N1) PB(N1) PC(N2)	
S1	N1		S1	N1	PC or PB	
S1	N1	PA				port doesn't offer srvc
S1	N1	PB	S1	N1	PB	
	N1		S1	N1	PC or PB	
	N1	PB	S1	N1	PB	

The translation rules, presented below, provide name translation for combinations of nodes, services, and ports.

3.1.2 Translation Process On A Source Node

To initiate a connection, a user presents to a source node a Resource Name consisting of: {SERVICE_NAME}{DESTINATION_NODE_NAME}{DESTINATION_PORT_NAME}. The DESTINATION_PORT_NAME presented without the DESTINATION NODE_NAME causes the source node to be chosen as a destination. If the DESTINATIO NODE_NAME is specified, the DESTINATION_PORT_NAME is optional.

At the SOURCE_NODE only the DESTINATION_NODE_NAME is translated/validated. If the DESTINATION_NODE_NAME is specified, no SERVICE_NAME translation is done and the DESTINATION_NODE_NAME is used to initiate a connection (if validation succeeded).

If no DESTINATION_NODE_NAME is specified, the DESTINATION_NODE_NAME is determined based on translation of the SERVICE_NAME from the advertised service database. The source node may reject the connection based on an unknown node, or, if a node isn't specified, based on the inability to translate the service to a node.

3.1.3 Translation Process On A Destination Node

The SERVICE_NAME is always translated on the DESTINATION_NODE. If the DESTINATION_PORT_NAME is specified, a port is selected and the service is selected at that port. If the service is not available at the specified port, the request is rejected with the "port doesn't offer service" reason. If the DESTINATION_PORT_NAME is not specified, an available port is selected by the DESTINATION_NODE for the requested service. If a service name is not specified on a source node and if a destination node does not have a default service (i.e., the default service is unnamed), the destination node may choose to accept or reject a connection. If the destination node accepts a connection, it translates an unspecified service name into a null-length name field. If the destination node rejects a connection, it returns the "no service available" reason.

The service name serves a dual role. The service name may be used to select a destination node at the source node, may be used to select a port at the destination node, and may be used to select the data processing function at the destination port (the latter describes the case of an application task running on a slave and not necessarily advertised as a service). LAT V5.1 provides a mechanism that allows a user to initiate a connection with an application task. In this case, another name (process name or task name) that uniquely identifies a task within a node is used as the service name.

Discussed three types of names are interpreted by the circuit and session layers as follows:

- NODE_NAME - Node names correspond to "service access points" (SAPs) or "sockets" in the host node virtual circuit layer. Each virtual circuit between a host node and a Terminal Server connects two of these virtual circuit layer service access points together. Source and destination node names together with source and destination Ethernet addresses uniquely identify each virtual circuit. The virtual circuit (CIRCUIT_NAME) on the master assumes the name of the host node (SLAVE_NODE_NAME). The virtual circuit (CIRCUIT_NAME) on the slave assumes the name of the server node (MASTER_NODE_NAME).

- SERVICE_NAME - Service names correspond to service access points in the slot layer in the host. These names are supplied to terminal server users for the purpose of providing a convenient means of identifying services. These names are especially useful in establishing slot sessions when more than one type of service is offered by a host node. The same service name might be specified by more than one host node if equivalent services are offered by the two different nodes. Destination service is specified by user is specified in the start slot.

- PORT_NAME - Port names correspond to physical or virtual ports on the host or server nodes. Connecting to the required service user can specify this name to direct data flow to the particular port.

3.2 Service Advertising Mechanisms

The following sections describe two methods of service advertising implemented by the LAT architecture.

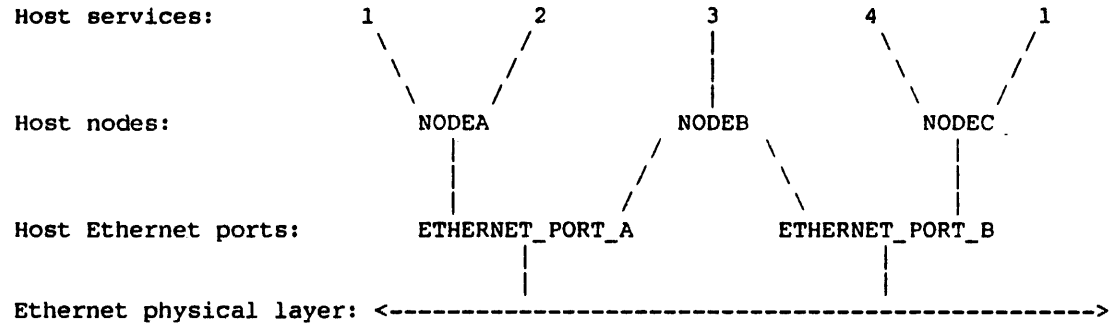
3.2.1 Host Advertising

To advertise availability of services, host nodes utilizes a single multicast message. This message enables the virtual circuit layer to translate node names into 48-bit Ethernet destination addresses and the slot layer to translate host service names into node names. The format and usage of this Service Class 1 message is described in the Service Class 1 appendix.

All messages multicasted by host nodes are required to specify a a node name of the host node. Systems that wish to announce an available service send multicast messages periodically.

In order to support the following environment:

Figure 3–4: Host Advertising



These multicast messages could be transmitted periodically:

Table 3–1: Multicasted Messages

Source Address:	PORT_A	PROT_A or PORT_B	PORT_B
Node Name:	NODE A	NODE B	NODE C
Service Names:	1 - rating 233 2 - rating 134	3 - rating 231	4 - rating 34 1 - rating 250

Which would cause the following database to be constructed in a server:

Table 3–2: Data Base on the Server

NODE_NAME	NODE_ ADDRESS	SERVICE_ NAME	SERVICE_RATING
NODEA	PORT_A	1 2	233 134
NODEB	PORT_A or PORT_B	3	231
NODEC	PORT_B	4 1	34 250

And results in the following display to a server user:

Service Name	Status	Description
1	Available	Description of service 1
2	Available	Description of service 2
3	Available	Description of service 3
4	Available	Description of service 4

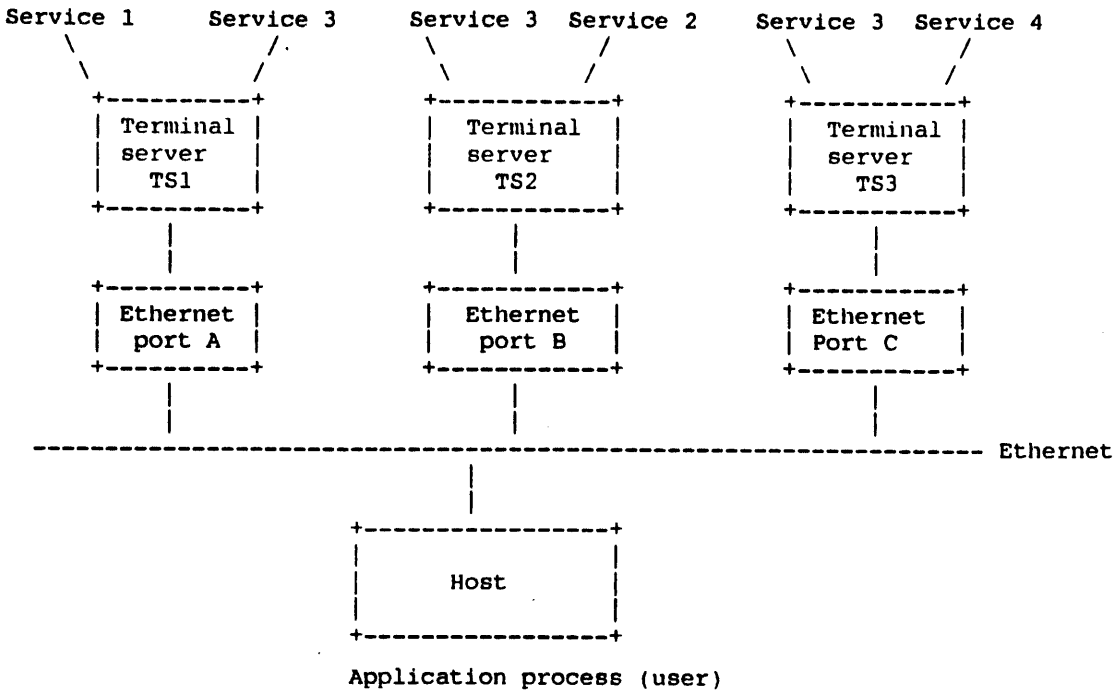
Note that the node names are not displayed to the user. If the user chooses service 1, the service rating is used to choose between the equivalent services.

3.2.2 Terminal Server Advertising

The service advertising mechanism eliminates the need for manual preparation of a database. To advertise services offered by the terminal servers two messages are used:

- Solicit information message (multicasted or physically addressed by a host node);
- Response message (physically addressed by the terminal server to the host node that sent Solicit message).

Figure 3–5: Server Advertising



Host node multicast information solicitation request for the Service 3. The response messages directed by all terminal servers to the host node will be:

Table 3–3: Information Available To The Host

Source Address	Port A	Port B	Port C
Node name	TS1	TS2	TS3
Service Information	Service 3 rating 100	Service 3 rating 43	Service 3 rating 57

The above messages allow the host node to process requested service information and choose the terminal server node to connect to. Type of processing may be chosen by the host node depending upon product requirements (building a full data base the way servers do, limiting number of entries in the data base, or processing without caching any data).

3.3 Specification of Names

The NODE_NAME, SERVICE_NAME, PORT_NAME and other architecturally specified names in the host multicast message, Start slot, and Start message are constrained to contain the following characters (refer to the DEC Multinational Character Set, STD 169):

- character code 2/4 ("$" - Dollar sign character).
- character code 2/13 ("-" - Hyphen or dash).
- character code 2/14 ("." - Period).
- character codes 3/0 to 3/9 ("0 through 9" - Numerals).
- character codes 4/1 to 5/10 ("A through Z" - Upper case letters).
- character code 5/15 ("_" - Underscore).
- character codes 6/1 to 7/10 ("a through z" - Lower case letters).
- character codes 12/0 to 15/15.

These names user are upcased before they are compared. Upcasing means subtracting the value 32 from the lower case letters in the range 6/1 through 7/10 and subtracting 32 from the character set in the range 14/0 through 15/15.

Notice that since these names are upcased before comparison, advertising lower case names is for display purposes only. If two different systems advertise the same name, one in lower case, and the other in uppercase, a number of different scenarios can result. Two typical scenarios are:

- If the NODE_NAME in the two messages is the same (upper and lower case), but the source address of the Ethernet packets are different, than the servers will detect this as a conflict and increment the DUPLICATE_NODE_NAME counter.
- If the NODE_NAME in the two messages is truly different, but one or more SERVICE_NAMES are the same (upper and lower case), the servers will "load balance" between the services.

If characters are specified in the international character set for service names, only those terminals that support the international character set will be able to select those host services.

Some implementations may not support the international character set. In local area networks that include these cases, the international character set should not be specified in names.

Note, that in the 5.1 version of the LAT protocol both - servers and hosts nodes are uniquely named. If a node name has not been supplied, a unique default node name must be created.

The physical address of each Ethernet port is guaranteed to be unique and can be used reliably to form a unique node name. The recommended procedure is to form the name from the human-readable form of the Ethernet address which is derived based on the algorithm described in the Ethernet specification document. The node name can be formed by combining the facility code and the human-readable form of the Ethernet address with the hyphens removed.

For example if the Ethernet address corresponds to the following sequence of bits on the Ethernet (bits are arranged from left to right):

0000 1111 0111 0100 1010 1000 0011 0110 1110 1110 1101 1001

the node name should be in the following human-readable form:

LAT_F02E156C779B

Note, that this unique node name must not be used to determine the Ethernet address. The real physical port address is present in the Ethernet message itself.

3.4 Specification of Text

Some messages contain fields representing textual or descriptive information. Valid characters present in these fields are 2/0 to 7/14 and 10/10 to 15/15.

4

Circuit and Session Layers

4.1 Architectural Model

This section presents state diagrams for the underlying virtual circuit (Virtual Circuit layer) and for slot session establishment (Slot layer).

Further discussion of many of the variables found in this section can be found in the "AXIOMS AND ALGORITHMS" section.

4.1.1 Slot Data

The term "slot data" means data supplied by any of the following functions:

- volunteer_xmt_data_a
- volunteer_xmt_data_b
- volunteer_attention_data
- queue_rcv_slot_buffer (creates a credit to be transferred)

4.1.2 Asymmetry

The host and terminal server state diagrams differ significantly. For this reason, they are presented separately. The state variables and mapping of received messages into the state diagrams are so similar that this material is presented once for the virtual circuit and once for the slot sessions. Frequently explicit notes point out that an item is relevant only to the host or to the terminal server.

4.1.3 Virtual Circuit Service

In order to establish a virtual circuit from a server to a host node, the Ethernet address, the host node name, and desired service class of the target host node must be known. The target host node name, Ethernet address and service class are usually determined from a multicast datagram received by the terminal server.

4.1.3.1 Virtual Circuit State

The state of a virtual circuit is captured in a data structure called the Circuit Block. The host and the terminal server maintain a separate Circuit Block. Changes to the Circuit Block are caused by events at the Ethernet port, events at the user interface and timers (counters) within the protocol state machine.

A general description of these variables follows. Algorithms for receiving and transmitting messages can be found in the "AXIOMS AND ALGORITHMS" section.

- CIRCUIT_NAME - the name of the virtual circuit
- REM_ADDRESS - the Ethernet address of the remote system.
- LOC_ADDRESS - the Ethernet address of the local system.
- MSG_TYP - Message type. The high order six bits of this field distinguish between different message types. The low order bit (bit 0) of this field is the RRF (response requested flag) flag. Bit 1 of this field is the the Master flag. It is always set in messages transmitted by the terminal server and always clear in messages transmitted by the host node.
- RRF - The Response Requested Flag. This bit is always clear in messages transmitted by the terminal server. This bit is conditionally set when messages are transmitted by the host node.
- REM_CIR_ID - Remote circuit identification
- LOC_CIR_ID - Local circuit identification (index to circuit block itself)
- NXMT - Next message number to transmit (modulo 256). This value is used to guarantee message sequencing. Every new message transmitted is numbered one higher than the previous message modulo 256 (254,255,0,1...).
- ACK - Highest message number received in sequence (modulo 256). This value is used to tell the session partner which sequenced message(s) have been received by the local system. It is transmitted in every message header for the remote session partner's use.

- DWF (terminal server only) - Data Waiting Flag. The flag is set by the virtual circuit layer whenever the RRF flag is set in a message received from the host node. DWF is also set by the slot layer whenever slot data is supplied.

  The DWF is cleared by the terminal server virtual circuit layer every time a new message is transmitted to the host that contains all of the available slot data supplied by the local users. Thus the DWF is cleared when all slot block Data Ready Flags are clear.

- DWF (host only) - Data waiting flag. This flag is set by the slot layer whenever any slot data is supplied. The DWF is cleared by the host virtual circuit layer every time a new message is transmitted to the terminal server that contains all of the slot data available from local users. Thus the DWF is cleared when all slot block Data Ready Flags are clear.

- LXMT - Lowest unacknowledged message number transmitted (modulo 256)

- HXMT - Highest unacknowledged message number transmitted (modulo 256)

- HOST_RETRANSMIT_TIMER (host only) - The host's retransmit timer is an interval timer which is started when the host node sends an "unsolicited" message to the terminal server. When this timer expires all unacknowledged messages are retransmitted.

- SERVER_CIRCUIT_TIMER (terminal server only) - This timer is used to initiate the transmission of new data but is not used to retransmit unacknowledged data.

- SERVER_RETRANSMIT_TIMER (Terminal server only) - This timer is used to retransmit unacknowledged messages. The terminal server retransmit policy is explained in the AXIOMS and ALGORITHMS section in the Message Transmitter section.

- HOST_RETRANSMIT_COUNTER, SERVER_RETRANSMIT_COUNTER - Count of number of times the current message number has been retransmitted. If this value reaches the LAT_MESSAGE_RETRANSMIT_LIMIT, one of two different policies can be enforced:

  1. The users of the circuit are notified that communications has been lost. The state of the virtual circuit is set to Halted.

  2. The users of the circuit are notified that communications has been temporarily interrupted. The state of the virtual circuit is not changed and messages continue to be retransmitted.

If the terminal server does not support multiple sessions, it is recommended that policy #1 be enforced. If the host crashes, policy #2 would "hang" all of the users until the host is rebooted and the terminal server transits the virtual circuit state through Halted; even users that attempt to disconnect would be "hung" in the disconnecting sub-state until a message was successfully transmitted and acknowledged or until the virtual circuit state reached Halted. If multiple sessions are supported, users can escape to a new virtual terminal.

- VC_QUALITY - A rating of the virtual circuit quality. Circuit quality is either acceptable or unacceptable.
- XMT_BUFFER_FREEQ - Linked list of available transmit buffers.
- UNACKED_XMTQ - Linked list of unacknowledged transmit messages. Messages are numbered from LXMT to HXMT consecutively, unless the queue is empty. (On the terminal server the length of this queue does not normally exceed one message.)
- SECONDS_SINCE_LAST_ZEROED (optional) - Seconds since the following counters were zeroed.
- MESSAGES_TRANSMITTED - Count of messages transmitted. The multicast messages transmitted by the host should be included in this total.
- MESSAGES_RECEIVED - Count of messages received.
- MESSAGES_RETRANSMITTED - Count of messages retransmitted because the message was not acknowledged.
- OUT_OF_SEQUENCE_MESSAGES_RECEIVED - Count of messages received which were not in sequence.
- ILLEGAL_MESSAGES_RECEIVED - Count of illegal messages received (see next section).
- ILLEGAL_SLOTS_RECEIVED - Count of illegal slots received (see next section).

4.1.3.2 Architecturally Controlled Names and Variables

4.1.3.3 Virtual Circuit State Variables

There are four state variables maintained by each end of a virtual circuit in the circuit block which are constrained by the architecture:

- LOC_CIR_ID - local circuit identification. This value is stored in the circuit block when the circuit block is created. The value zero is reserved and is not

valid as a LOC_CIR_ID. Each valid Run message received by the local system will have the DST_CIR_ID field of the received message equal to the LOC_CIR_ID field in some circuit block.

The LOC_CIR_ID value should be defined by the system to help locate the circuit block. If a virtual circuit to a partner should fail, and a new circuit to the same partner is to be formed, the values of LOC_CIR_ID used to form the new circuit must be different than the value used in the previous circuit. Normally this is accomplished by using a sequence number.

- REM_CIR_ID - remote circuit identification. Initially the REM_CIR_ID value is zero in the circuit block. The source of this value is the SRC_CIR_ID field in received messages. This non-zero value references the remote system circuit block.

- NXMT - next message number to transmit. This circuit block value is a modulo-256 value that is used to assign the message header field MSG_SEQ_NBR.

- ACK - the number of the most recent message received in sequence. This circuit block value is also a modulo-256 value. It is copied from the MSG_SEQ_NBR field of any message received in sequence (including Start messages) to the circuit block ACK field. Every time a message is transmitted, this value is copied into the message MSG_ACK_NBR field.

4.1.3.4 Slot State Variables

The LOC_SLOT_ID is a value assigned by the local implementation. Received Run slot DST_SLOT_IDs will be identical to the value transmitted in the Start slot SRC_SLOT_ID field used to establish the slot session. For this reason, the value should be assigned as an index into an array of slot block addresses. The value LOC_SLOT_ID is constrained to be non-zero.

REM_SLOT_ID is a value stored in the local slot block which is used to validate received Run slots. Initially the REM_SLOT_ID is zero in the slot block. The source of this value is the SRC_SLOT_ID field in received Start slots. This non-zero value references the remote system slot block.

4.1.3.5 Message Counters

The implementor of Digital products should read "Digital Ethernet Node Product architecture" specification. This document specifies generic product requirements for Digital Ethernet nodes.

The purpose of requiring counters is to identify hardware faults and software implementation errors.

Required counters must be displayable on demand by a privileged user on a terminal connected to the local system. The description of the displayed counters must resemble the descriptions used in this section.

These counters should be zeroed as infrequently as possible. Ideally, counters should be zeroed by command of a privileged user only. It may be desirable for nonprivileged users to be able to display counters.

If a virtual circuit to a remote system is halted, the associated counters must must not be zeroed (although they may be deleted). An implementation should attempt to retain counters even after communications with a remote system has terminated so long as this requirement causes only idle resource consumption.

Values must be unsigned 32-bit integers. The values must "latch" the highest possible value if they overflow.

An implementations must collect and be capable of locally displaying the following list of values:

- Those required in "Digital Ethernet Node Product Architecture"
- Total number of Illegal messages received (and associated Ethernet physical address if possible)
- Total number of Illegal slots received (and associated Ethernet physical address if possible)

NOTE

Illegal message and slot counters are maintained as a part of the virtual circuit database listed above. If these counter databases are retained for some time after the virtual circuits are terminated, a valuable piece of information is retained to diagnose the source of the illegal data - the Ethernet physical address the remote system | A minimum of one circuit block must be retained by an implementation after active circuits are terminated. This is to retain the circuit block counters for some minimum amount of time for error diagnosis.

Implementations must collect and be capable of locally displaying the following list of values for each currently active virtual circuit:

- (best effort) SECONDS_SINCE_LAST_ZEROED (optional)

- MESSAGES_TRANSMITTED
- MESSAGES_RECEIVED
- MESSAGES_RETRANSMITTED
- OUT_OF_SEQUENCE_RECEIVED
- ILLEGAL_MESSAGES_RECEIVED (causing the virtual circuit to be aborted)
- ILLEGAL_SLOTS_RECEIVED (causing the virtual circuit to be aborted)

Remember that messages are Ethernet frames under virtual circuit control. Multicast datagrams are not messages.

4.1.3.6 Error Handling - Illegal Slots And Messages

Virtual circuits are immediately stopped when illegal messages or slots are received.

Illegal messages and slots are those that do not conform to the defined message formats or grossly violate the defined state transitions of a virtual circuit or user slot sessions. These messages and slots should not occur, but if they do:

- Minimally, a displayable counter must be incremented. Ideally, the affected users and the system manager should be immediately notified of this unusual event.
- As much of the message or slot as possible should be stored in order to diagnose the failure. Optionally store the number of the error (see below).
- The message should be discarded.
- The (underlying) virtual circuit must be stopped.

The term "illegal" should be distinguished from the term "invalid". Invalid messages and slots are normal events and are usually caused by improper synchronization (resynchronization of virtual circuits and user sessions).

Examples of illegal messages are:

1. A received message with either the DESTINATION_ADDRESS or the SOURCE_ADDRESS equal to zero.
2. An unknown MSG_TYPE in a received message.

3. A non-zero SRC_CIR_ID in a received Stop message.
4. A zero SRC_CIR_ID in a Start or Run message.
5. A non-zero DST_CIR_ID in a Start message received by a host node.
6. A zero DST_CIR_ID in a Run or Stop message sent by the terminal server.
7. A zero DST_CIR_ID in a Start, Run or Stop message sent by the host.
8. Others - please get them added to this list.

Examples of illegal slots are:

1. An unknown SLOT_TYPE value is received.
2. An unknown SERVICE_CLASS is received in a Start slot.
3. A non-zero SRC_SLOT_ID in a received Stop slot.
4. A zero SRC_SLOT_ID in a Start slot.
5. A non-zero DST_SLOT_ID in a Start slot received by a host node.
6. A zero DST_SLOT_ID in any slot but a Start slot sent by the terminal server.
7. An Attention slot specifying a non-zero value in the credit field (documented as an MBZ field in the message format section).
8. a Start slot received in the Run state (without and intervening Stop slot)
9. a Reject slot received in the Run state.
10. a Data_a or Data_b slot arrives which contains data (consumed a remote credit), but no user buffer is available (no credit was extended).
11. a Run slot with a zero SRC_SLOT_ID.
12. Others - please get them added to this list.

4.1.3.7 Defined Parameters And Recommended Or Required Default Values

Some of the values defined in this section may be changed by the system manager. The name of the variable should be similar to the name used below in command interfaces.

The follow values are architecturally defined constants:

- Protocol Type - 60-04 (hexadecimal) or as a bit stream, first bit on Ethernet at left (0000.0110.0010.0000)
- Multicast Address - 09-00-2B-00-00-0F (hexadecimal) or as a bit stream, first bit on Ethernet at left (1001.0000.0000.0000.1101.0100.0000.0000.0000.0000.1111.000)

The following values must be specified before an implementation is operational. If an implementation allows the values are settable within the ranges specified, the names used to refer to the parameters must be a reasonable facsimile of the name used below. If the parameters are not settable, the recommended default values should be used. The architecture requires the values be within the ranges specified:

- PROTOCOL_VERSION - The value 5.
- PROTOCOL_ECO - The value 1.
- SERVICE_NAME - must be specified by the host system manager before the start of service can be announced.
- NODE_NAME - must be specified by the host system manager before the start of service can be announced.
- SERVER_CIRCUIT_TIMER - In the range 1-100. This parameter specifies 10 millisecond intervals. (The value 8 is recommended - 80 milliseconds).
- SERVER_RETRANSMIT_TIMER - In the range 1 to 2 seconds.
- HOST_RETRANSMIT_TIMER - In the range 1 to 2 seconds.
- LAT_MESSAGE_RETRANSMIT_LIMIT - Four or more messages. Eight messages recommended for the terminal server (SERVER_RETRANSMIT_ COUNTER limit), more than 60 messages recommended for the host node (HOST_RETRANSMIT_COUNTER limit).
- HOST_MULTICAST_TIMER - In the range 10 to 180 seconds (60 seconds recommended). This value is supplied via the start_service_class function.
- LAT_MIN_RCV_DATAGRAM_SIZE - In the range 576 to 1518 (1518 recommended) for both the host node and terminal server.
- LAT_MIN_RCV_SLOT_SIZE - In the range 1 to 255 (127 recommended) for both the host and terminal server.

- LAT_MIN_RCV_ATT_SLOT_SIZE - In the range 1 to 255 (31 recommended) for both the host and terminal server.
- NBR_DL_BUFS - The number of data link buffers assigned minus one. A value of zero is recommended.
- A Host implementation should enable multicast group 0 by default (unless specified differently interactively).
- A Server implementation should enable multicast group 0 by default (unless specified differently interactively).
- RESPONSE_TIMER - in the range 1 to 2 seconds.
- MULT_STAT_TIMER - in the range 10 to 100 milliseconds.
- STAT_REP_TIMER - in the range 10 seconds to 1 hour. The recommended value is 60 seconds.
- RETR_COMM_TIMER - is recommended to be 1 second.
- RETR_COMM_COUNT - is recommended to be a value of 3 or 4.
- RETR_STAT_TIMER - is recommended to be 1 second.
- RETR_STAT_COUNT - is recommended to be a value of 3 or 4.
- PRODUCT_TYPE_CODE -

  1. Ethernet terminal server
  2. Decserver 100
  3. VAX/VMS
  4. RSX11-M
  5. RSX11-M+
  6. TOPS-20
  7. TOPS-10
  8. ULTRIX-11
  9. LAT-11
  10. RSTS/E
  11. ULTRIX-32

12. ELN
13. MS/DOS
14. P/OS
15. PCSG-LAT
16. DELIX
17. DECserver 200
18. DECserver 500
19. Actor

- PRODUCT_VERSION_NUMBER - version number of a product.

The following value must be supplied before an implementation is convenient to use:

- LAT_KEEP_ALIVE_TIMER - In the range 10 to 255. A value of 20 seconds is recommended.

The parameters that are optional are:

- FACILITY_NUMBER - A value supplied via the local command interface.
- SERVER_NAME - A name supplied via the local command interface.
- LOCATION_TEXT - Text supplied via the local command interface.
- Parameter data supplied by an implementation's software modules or via the local command interface. See individual service classes for a description of these parameters.

4.1.3.8 Message Types

There is a common virtual circuit header format for LAT messages documented in the section "MESSAGE FORMATS" (other message formats are defined by the different service classes). This virtual circuit message format is one of three different types:

1. start message - Start messages are used to establish new virtual circuits.
2. run message - Run messages convey state and slot data between systems.

3. stop message - Stop messages are used to end a virtual circuit session.

A stop message is also used as a "no circuit" message to reply to a received message which references a nonexistent or invalid virtual circuit (see state diagrams "Send Stop"). An implementation should make a best effort to send these "no circuit" messages. Occasionally, due to lack of resources or Ethernet data link errors, these "no circuit" messages may fail to get delivered. Failure to send these "no circuit" messages can result in slow resynchronization after the host node or server crashes.

4.1.3.9 Cirtual Circuit State Variables

There are three virtual circuit states: Halted, Starting and Running. Once the system to system virtual circuit has started successfully, the circuit reaches the Running state.

These events determine state transitions:

1. VC_start (server only) - user requests virtual circuit startup. An implementation would allocate a Circuit Block at this point if none existed.
2. VC_halt - user requests that the virtual circuit be halted immediately
3. Start_rcv - Start message received. An implementation would allocate a Circuit Block at this point, if one did not exist from a previous circuit, and initialize all state variables.
4. Inv_start_rcv - invalid Start received (see message mapping section)
5. Stop_rcv - Stop message received.
6. Inv_stop_rcv - invalid Stop received (see next section)
7. Run_rcv - Run message received with valid connection identification. The message is in sequence if MSG_SEQ_NBR of received message equals the value ACK+1 in the circuit block (modulo 256).
8. Inv_run_rcv - invalid Run received (see next section)
9. Circuit_timer - the SERVER_CIRCUIT_TIMER expires.
10. Rexmit_timer - The SERVER_RETRANSMIT_TIMER or HOST_RETRANSMIT_TIMER expires.
11. Resend_limit - The retransmit counter reached the limit LAT_MESSAGE_RETRANSMIT_LIMIT.

12. Send_data (host only) - A user supplies slot data and the RRF flag in the circuit block is clear. Note that this event is blocked if the RRF is set.

4.1.3.10 Response Requested Flag and Balanced Mode

When the most recent message received from the host node has the RRF clear (no response requested), and this message acknowledges the last message transmitted from the terminal server, the virtual circuit is said to be "balanced". When in this state, the host node has permission to send one "unsolicited" message and the terminal server will not send messages if the DWF (data waiting flags) is clear.

This state will persist until either the Send_data event occurs in the host or the DWF is set in the terminal server (possibly due to the keep alive timer).

"Balanced mode" prevents the LAT protocol from consuming unnecessary Ethernet bandwidth. Without balanced mode, LAT messages would be exchanged at SERVER_CIRCUIT_TIMER millisecond intervals, even though no useful data was being exchanged.

There is a sub-state that is not evident from a cursory inspection of the state diagrams. This sub-state is entered when the event Send_data occurs. Notice that this event causes the HOST_RETRANSMIT_TIMER to be started and may cause the Circuit Block RRF flag to be set. Also notice that this event cannot occur if the RRF flag is already set. This sub-state retransmits all unacknowledged messages whenever the HOST_RETRANSMIT_TIMER expires. This sub-state is exited when the host node receives a message that acknowledges all of the currently unacknowledged messages. At this point and the HOST_RETRANSMIT_TIMER is stopped. The purpose of this sub-state is to guarantee "unsolicited" message delivery.

4.1.3.11 Message Mapping Onto State Diagram

Received messages must be validated.

The Ethernet data link layer verifies that the Ethernet DESTINATION_ADDRESS field in the received message matches the address assigned to the local system and that the PROTOCOL_TYPE field in the received message is equal to the LAT protocol type.

On the slave end the LAT virtual circuit layer maps Start messages onto circuit blocks based on the value of the MASTER_NODE_NAME field and the SOURCE_ADDRESS field of the received Start message. A match to a circuit block is found if the MASTER_NODE_NAME field of the received Start message equals the CIRCUIT_NAME field in the circuit block AND the SOURCE_ADDRESS field of the received Start message equals the REM_ADDRESS field in the circuit block.

A Start message can match a circuit block even if it is in the Running state (as shown in the state diagrams).

On the master end the LAT virtual circuit layer maps Start messages onto circuit blocks based on the value of the DST_CIR_ID. The SLAVE_NODE_NAME field and the SOURCE_ADDRESS field of the received Start message must match to the CIRCUIT_NAME field AND the REM_ADDRESS field in the referenced circuit block.

In the case of the first Start_rcv event in the host node, no circuit block will exist to reference. A circuit block should be allocated and the state variables should be initialized as described below the host virtual circuit state table. The DST_CIR_ID of the received message should be considered a match to the LOC_CIR_ID of the circuit block in this case. If a circuit block cannot be allocated, the implementation should attempt to send a Stop message that indicates no resources.

Note that it is possible for a pair of nodes that each implement terminal server and host capabilities (master and slave capabilities) to establish exactly one virtual circuit in each direction, but not more.

Run messages and Stop messages are mapped onto a circuit block based solely on the value of the DST_CIR_ID field of the received message. The value DST_CIR_ID must match the value LOC_CIR_ID in the referenced circuit block and the value SRC_CIR_ID must match the REM_CIR_ID value in the circuit block. Run messages that do not map onto a circuit block in the Running state are discarded and a Stop message is sent to the remote system.

Next, the message type is determined from the LAT header MESSAGE_TYPE field. The received message is then mapped into the state diagram based on the following rules:

- Start_rcv - The DST_CIR_ID of the received message must equal the LOC_CIR_ID in the referenced circuit block (except as noted above). The SRC_CIR_ID of the message must not be zero, and is copied to the REM_CIR_ID of the referenced circuit block.
- Inv_start_rcv - The DST_CIR_ID of the received message is non-zero and is not equal to the LOC_CIR_ID in the referenced circuit block.
- Run_rcv - The DST_CIR_ID of the received message equals the LOC_CIR_ID of the circuit block, and the SRC_CIR_ID of the received message equals the REM_CIR_ID of the circuit block. The message is in sequence if the MSG_SEQ_NBR of the received message equals the value ACK+1 in the circuit block (modulo 256).

- Inv_run_rcv - The DST_CIR_ID of the message is not equal to the LOC_CIR_ID in the circuit block, or the SRC_CIR_ID of the message is not equal to the REM_CIR_ID of the circuit block.
- Stop_rcv - The DST_CIR_ID of the message equals the LOC_CIR_ID of the circuit block, and the SRC_CIR_ID of the message equals zero.

Invalid messages are the result of improper synchronization between the host node and terminal server. These events are normal; the message is treated as described in the state diagrams.

4.1.3.12 Terminal Server Virtual Circuit State Table

Table 4–1: Terminal Server Virtual Circuit State Table

State	Event(s)	Action(s)	Next State
Halted	VC_start	Initialize, Send Start.	Starting
	Stop_rcv	No action.	Halted
	Inv_stop_rcv	No action.	Halted
	any other msg	Process Start, Send Stop.	Halted
	any other	No action.	Halted
Starting	Start_rcv	Process Start, send Run. (typically including Start slot)	Running
	Resend_limit	Notify users, send Stop.	Halted
	Rexmit_timer	Resend Start.	Starting
	Stop_rcv	Notify users.	Halted
	VC_halt	Send Stop	Halted
	Inv_stop_rcv	No action.	Starting
	any other msg	Process Start, send Start.	Starting
Running	Run_rcv	If msg is out of sequence: zero NBR_SLOTS in msg hdr. If RRF flag is set: set DWF. Process received ack; process message.	Running
	Rexmit_timer	If messages remain unacknowledged: resend all unacked messages.	

Table 4–1 (Cont.): Terminal Server Virtual Circuit State Table

State	Event(s)	Action(s)	Next State
Running (cont.)	Circuit_timer	If messages acked and DWF set: send message; clear DWF if all slot data has been sent. If messages acked and DWF clear: no action.	Running
	Resend_limit	Notify users (or optionally halt via VC_halt event).	Running (Halted)
	VC_halt	Send Stop	Halted
	Stop_rcv	Notify users	Halted
	any other	set DWF	Running

Note that the server slot state table also specifies that DWF be set.

Terminal Server Virtual Circuit State Table Notes

"Process Start" means copy the SRC_CIR_ID field from the received Start message to the REM_CIR_ID field in the circuit block. Verifies circuit block by matching SLAVE_NODE_NAME and the SOURCE_ADDRESS from the start message against CIRCUIT_NAME and REM_ADDRESS.

The "Initialize" means the values in the Circuit Block are initialized as:

1. CIRCUIT_NAME<- <the name passed by the VC_start function>
2. REM_ADDRESS<- <value passed in the VC_start call>
3. LOC_ADDRESS<- <value assigned to the local system>
4. REM_CIR_ID<- 0 (later copied from received Start message)
5. LOC_CIR_ID<- <unique virtual circuit connection id>
6. NXMT<- 0 - Next message number to transmit
7. ACK <- 255 - message number most recently received in sequence
8. LXMT<- 0 - Lowest unacknowledged message number transmitted
9. HXMT<- 0 - Highest unacknowledged message number transmitted

10. SERVER_CIRCUIT_TIMER<- <reset to ~ 80 ms> (count-down to zero)

11. SERVER_RETRANSMIT_COUNTER<-0 (count-up to LAT_MESSAGE_REXMIT_LIMIT)

12. UNACKED_XMTQ<- <empty>

13. RRF and DWF are cleared

4.1.3.13 Host Virtual Circuit State Table

Table 4–2: Host Virtual Circuit State Table

State	Event(s)	Action(s)	Next State
Halted	Start_rcv or Inv_stop_rcv	Initialize; process Start; Send Start (or Stop).	Starting (or Halted)
	Stop_rcv or Inv_stop_rcv	No action.	Halted
	any other msg, or no resources	Process Start; send Stop.	Halted
Starting	Run_rcv	If msg is out of sequence: zero NBR_SLOTS in msg header. Process rcvd ACK. Process rcvd message.	Running
	VC_halt	Send Stop	Halted
	Stop_rcv	No action.	Halted
	Inv_stop_rcv	No action.	Starting
	Start_rcv	Initialize; process Start; send Start	Starting
	any other msg	send Start	Starting
Running	Run_msg_rcv	If msg is out of sequence: zero NBR_SLOTS in msg header. Process rcvd ACK; if all msgs are acked: stop timer. Process rcvd message; queue_transmit_message; transmit_unacknowledged_queue.	Running

Table 4-2 (Cont.): Host Virtual Circuit State Table

State	Event(s)	Action(s)	Next State
Running (cont.)	Send_data	Start timer; queue_ transmit_message; transmit_unacknowledged_ queue.	Running
	Rexmit_Timer	Resend unacked msgs; reset timer.	Running
	Resend_limit	Notify users (or optionally halt via VC_halt event).	Running (Halted)
	VC_halt	Send Stop.	Halted
	Stop_rcv	Notify users.	Halted
	Start_rcv	Initialize; process Start; send Start	Starting
	any other	Transmit_unacknowledged_ queue.	Running

"Process Start" means copy SRC_CIR_ID from the received Start message into the circuit block field REM_CIR_ID; copy the SRC_ADDRESS from the message into the REM_ADDRESS field in the circuit block and copy the MASTER_NODE_ NAME from the received Start message into the circuit block CIRCUIT_NAME field.

"Initialize" means the values in the Circuit Block are initialized as:

1. REM_ADDRESS<- <SOURCE_ADDRESS from the received Start message>
2. LOC_ADDRESS<- <value assigned to the local system>
3. REM_CIR_CID<- <copied from received Start message>
4. LOC_CIR_ID<- <unique connection id>
5. NXMT<- 0 - Next message number to transmit
6. ACK <- 0 - most recent message number received in sequence
7. LXMT<- 0 - Lowest unacknowledged message number transmitted
8. HXMT<- 0 - Highest unacknowledged message number transmitted
9. HOST_RETRANSMIT_TIMER<- <stopped>

10. HOST_RETRANSMIT_COUNTER<- 0 (counts up to LAT_MESSAGE_REXMIT_LIMIT)

11. UNACKED_XMTQ<- <empty>

12. RRF flag is cleared

13. DWF is cleared

"Start rexmit_timer" starts a one to two second interval timer. This timer is used to retransmit messages that do not get acknowledged by the terminal server within the expected time limit. One second is arbitrarily larger than the timer value used by the terminal server (SERVER_CIRCUIT_TIMER). If this HOST_RETRANSMIT_TIMER expires, most likely the terminal server has crashed or the Ethernet data link has failed.

4.1.4 User Connection Management And Data Flow

Variables in this section are described in sufficient detail to explain the slot state transitions necessary to establish and maintain slot sessions.

4.1.4.1 Service Classes

It is the responsibility of the slot layer to deliver start slots (connect requests) to the appropriate service class in the host. Each service class has the freedom to define the capabilities utilized by that service class independently. For instance, the service class A might utilize attention slots while the service class B might not.

After the start slot has been accepted by the service class, and a start slot sent in response, all subsequent slots associated with that session are delivered to the same service class.

Therefore the virtual circuit service can be shared by one or more service classes, while each service class utilizes different slot types and slot formats.

4.1.4.2 Host Session Management

The host implementation may choose to always accept or reject a connection (respond to a start slot received by the host service with a start or reject slot) based solely on currently available resources. (Lack of resources might include such criteria as nonavailability of memory, too many users or system shutdown in progress). This type of implementation is appropriate if the connect request contains insufficient information for the host service to make a decision to reject the connection. Interactive terminal login is an example of such a host service. The account name and password must be supplied within the context of the session before the host service can reject the session.

Alternatively, a host implementation may offer the host service a chance to reject or accept directly. This is the behavior modeled in the host slot state table.

4.1.4.3 Multiplexing Over A Virtual Circuit

To provide multiple users connection management and data transfer service simultaneously, the underlying virtual circuit can be shared by all active users. This sharing is accomplished by dividing each message into a header and one or more slots. Whenever more than one user has volunteered data to be transmitted, an individual user's ability to transmit data is restricted by the following rules, which guarantee each user gets treated fairly:

- limiting the slot size - the maximum slot size is the slot header size (4 bytes) plus the maximum data size (255 bytes), or 259 bytes. Thus one 1518 byte message has enough room for at least five, and usually more slots.
- slot flow control - Data_a and Data_b slots cannot be transmitted unless a transmit slot credit (LOCAL_CREDITS) is owned by the user.
- limiting each user to one slot until all other users have been considered
- not starting with same user each time - if not all of the slot data fit into a message, the next scan for slot data should resume with the users that did not get slot data into the message the previous time.

As with frames on the Ethernet, slots are divided into a header and a data section. Connection management is accomplished by defining the state transitions of the slot headers as described in the following sections.

4.1.4.4 Slot Ordering Within Messages

Slot ordering within a message is not arbitrary. If two or more slots in a buffer are addressed to the same user, the slots are processed from the beginning of the buffer to the end.

For instance, an "abort" slot received by a terminal server will abort any output data in slots received before it within the message (and in previous messages), but will not affect a slot in the buffer following it.

4.1.4.5 Slot State Variables

The state transitions for slots are similar to those for the underlying virtual circuit itself. Any of the slot transitions shown in the slot state tables assume an underlying virtual circuit in the Running state. If the virtual circuit should exit the Running state, the slot sessions immediately transfer to the halted state.

The state of a slot session is captured by each end in a data structure called the Slot Block. Changes to the Slot Block are caused by events at the Ethernet port and events at the user interface.

The virtual circuit layer delivers to the slot layer messages that contain one or more slots. Slot validation is the responsibility of the slot layer.

The slot block contains the following fields:

- REM_SLOT_NAME - the name of the remote slot block. Required to be the name of the remote service selected.
- LOC_SLOT_NAME - the name of the local slot block
- REM_SLOT_ID - Remote slot connection identification
- LOC_SLOT_ID - Local slot connection identification
- REMOTE_CREDITS - credits being extended to the session partner. This credit total is zeroed by the slot layer each time the slot multiplexer copies the slot into a message buffer. This total is incremented every time the user adds a receive buffer via the queue_rcv_slot_buffer function.
- LOCAL_CREDITS - available credits to transmit slots. This field is initialized to zero when the slot block is created. The slot layer slot demultiplexer adds any credits extended in the received slot CRED field to this slot block LOCAL_CREDITS field. The slot layer slot multiplexer decrements this field whenever a Data_a or Data_b slot is copied into a message buffer with a non-zero SLOT_BYTE_COUNT. Data_a and Data_b slots do NOT consume credits if the SLOT_BYTE_COUNT is zero (to prevent infinite looping |). Start, Stop and Attention slots do not consume credits.
- SLOT_TYPE - slot type. One of Start, Data_a, Data_b, Attention, Reject or Stop.
- DRF - Data Ready Flag. Set whenever slot data is available. Cleared by the slot layer whenever all slot data is under virtual circuit control.
- SLOT_COUNT - byte count of next field (which could be zero)
- SLOT_DATA_BUFFER - This buffer is used to store Data_a received over the session as the result of extending slot credits.
- ATTENTION_DATA_BUFFER - This buffer is used to deliver Attention data to the user. A semaphore (not shown in the slot block) is used to arbitrate ownership of the buffer. Since Attention data is not flow controlled, Attention data is discarded if new data is delivered and the buffer is not available.

These events determine state transitions:

1. Connect_req (terminal server only) - user requests connection.
2. Disconnect_req - user requests disconnection.
3. Reject_req (host only) - user rejects a requested connection.
4. Accept_req (host only) - user accepts a requested connection.
5. Start_rcv - start slot received.
6. Stop_rcv - stop slot received .
7. Reject_rcv - reject slot received.
8. Run_rcv - Data_a, Data_b or Attention slot received.
9. User_data - user supplies data via volunteer_data_a or volunteer_data_b.
10. User_rcv - user supplies receive slot buffer via queue_rcv_slot_buffer.
11. Stop_sent - Stop slot under virtual circuit control.

4.1.4.6 Terminal Server Slot Mapping Onto State Diagram

Received slots are mapped onto the events based on the values received in the received slot DST_SLOT_ID and SRC_SLOT_ID fields, and the current slot block values of REM_SLOT_ID and LOC_SLOT_ID.

The key for the symbols in the table:

- L - DST_SLOT_ID from the received slot equals LOC_SLOT_ID in the referenced slot block
- R - SRC_SLOT_ID from the received slot equals REM_SLOT_ID in the referenced slot block
- I - DST_SLOT_ID from the received slot does not equal LOC_SLOT_ID in the referenced slot block
- J - SRC_SLOT_ID from the received slot does not equal REM_SLOT_ID in the referenced slot block

- D - Doesn't matter what SRC_SLOT_ID in the received slot is
- 0 - DST_SLOT_ID or SRC_SLOT_ID in received slot is zero

DST_SLOT_ID	SRC_SLOT_ID	State	Type of slot or action to be taken
L	R	Running	Data_a, Data_b or Attention slot
L	D	Starting	Start slot (D cannot be zero)
L	0	any	Stop or Reject slot
L	J	Running	send Stop to SRC_SLOT_ID
I	D	Starting	ignore slot (session shutting down)
I	D	Running	ignore slot (session shutting down)

Events shown in this list but not in the table below are either invalid or are events that occur as a session shuts down. The list suggests an appropriate action.

4.1.4.7 Terminal Server Slot State Table

Table 4–3: Terminal Server Slot State Table

State	Event	Action	Next State
Halted	connect_req	Initialize, Send Start.	Starting
	any other	No action.	Halted
Starting	disconnect_req	No action.	Abort_start
	Reject_rcv	No action.	Halted
	Start_rcv	process SRC_SLOT_ID; update credits; process Start.	Running
	any other	No action.	Starting
Abort_strt	Reject_rcv	No action.	Halted
	Start_rcv	Send Stop	Halted
	any other	No action.	Abort_start
Running	disconnect_req	Send Stop (see note below).	Stopping
	Stop_rcv	Notify user.	Halted

Table 4-3 (Cont.): Terminal Server Slot State Table

State	Event	Action	Next State
Running (cont.)	Reject_rcv	Illegal slot (Declare VC_ halt)	Halted
	Start_rcv	Illegal slot (Declare VC_ halt)	Halted
	Run_rcv	Update credits and process slot.	Running
	User_data or User_rcv	Set DWF and DRF.	Running
	any other	No action.	Running
Stopping	Stop_sent	No action.	Halted
	any other	No action.	Stopping

A Start_rcv event need not be distinguished from a Run_rcv except in the Starting state. In the Starting state the Start slot should be validated based on Slot_type to be sure a Run slot is not being received.

"process SRC_SLOT_ID" means copy the SRC_SLOT_ID field from the received slot into the REM_SLOT_ID field of the slot block.

"Update credits" means add the CREDIT field in the received slot into the LOCAL_ CREDITS field of the slot block.

The "Send stop" action in the table may involve queue delays until the virtual circuit layer accepts the Stop slot. Until the Stop slot is accepted under virtual circuit control, all other received messages should be ignored, and the transition to the Halted state must be delayed. This prevents the connect_req event from reusing a slot state table until a queued stop has been accepted by the virtual circuit layer.

4.1.4.8 Host Slot Mapping Onto State Diagram

Slots are mapped onto the events based on the values received in the slot DST_ SLOT_ID and SRC_SLOT_ID fields, and the current slot block state variables LOC_ SLOT_ID and REM_SLOT_ID. The key for the symbols in the table:

- L - DST_SLOT_ID from the received slot equals LOC_SLOT_ID in the referenced slot block
- R - SRC_SLOT_ID from the received slot equals REM_SLOT_ID in the referenced slot block

- I - DST_SLOT_ID from the received slot does not equal LOC_SLOT_ID in the referenced slot block
- J - SRC_SLOT_ID from the received slot does not equal REM_SLOT_ID in the referenced slot block
- D - Doesn't matter what SRC_SLOT_ID in the received slot is
- 0 - DST_SLOT_ID or SRC_SLOT_ID in received slot is zero

DST_SLOT_ID	SRC_SLOT_ID	Type of slot or action to be taken
L	R	Data_a, Data_b or Attention slot
L	J	Ignore (stop slot followed by reassignment of LOC_SLOT_ID.)
L	0	Stop or Reject slot
0	D	Start (D cannot be zero).
I	D	ignore slot (session shutting down)

The Start_rcv event <0 D> can occur in any state, but the event is always mapped onto a new slot block in the halted state.

Events shown in this list but not in the table below are either invalid or are events that occur as a session shuts down. The list suggests an appropriate action.

4.1.4.9 Host Slot State Table

Table 4–4: Host Slot State Table

State	Event	Action	Next State
Halted	start_rcv	Init; request session of user.	Starting
	Any other	No action.	Halted
Starting	reject_req	Send Reject.	Halted
	accept_req	Send Start.	Running
	Any other	No action.	Starting
Running	Disconnect_req	Send Stop.	Stopping

Table 4–4 (Cont.): Host Slot State Table

State	Event	Action	Next State
Running (cont.)	Stop_rcv	No action.	Halted
	Run_rcv	Process slot.	Running
	User_data or User_rcv	Set DWF and DRF.	Running
last			
	Any other	No action.	Running
Stopping	Stop_sent	No action.	Halted
	Any other	No action.	Stopping

A host implementation may choose to always accept a requested session. In this case the Starting state in the table would not exist. The start_rcv event would establish a session (assuming sufficient resources). A host service (the terminal class driver) might then later reject the session via the disconnect_req. This example would occur during a terminal server user login failure.

4.2 Layer Interfaces

The interfaces presented in this section are not meant to be implemented. The purpose is to present the control and data flowing through the LAT layers between the users in a way that unambiguously describes what is required at the interfaces, but allows implementations the freedom necessary to implement the functions appropriately for each different system.

The model presented implies that each side of the interface both provides service entry points and utilizes service entry points provided to it. Synchronization across interfaces is the responsibility of the implementation. Specifically, the implementation must assure that all user and Ethernet data link interface events are processed serially and atomically, in both directions. This requirement arises primarily from the need to maintain predictable states in the shared Circuit and Slot blocks.

The polling model is used to show correspondence of functions and to reduce the amount of text necessary to describe the interfaces. Only one interface is described at each layer. In fact, implementations would offer only a subset of the functions described, one subset corresponding to the terminal server and another corresponding to the host.

In interface calls:

- input parameters are specified first
- input parameters are separated from output parameters by a semicolon
- parameters are separated by commas
- "S-","H-", and "SH-" indicate that a function is available at the terminal server ("S-"), host ("H-") or both ("SH-").

Within the interface calls, the "reason" argument is defined separately for each different service class (see the appendices).

4.2.1 Data Types

There are two distinct types of data that can be transferred between session partners: Data streams (Data_a and Data_b) and attention data. The data streams are flow controlled and error controlled. The attention data is error controlled, but is not flow controlled. Idempotent operations and data not requiring delivery can be transferred in attention slots.

The purpose of having both Data_a and Data_b data streams is:

- Status and control information (Data_b) can be transferred as a separate data stream. An implementation does not have to embed the status and control information in the data stream or operate a separate virtual circuit.
- Status and control (Data_b) transfers are phase locked relative to Data_a transfers. If the Data_b slot indicates a change is to be applied to the Data_a stream, the change is unambiguously applied to the subsequent Data_a slots.
- The implementation is free to define the entire format of the Data_b slots. This provides much greater flexibility since the Data_a channel is normally unformatted.

The rules governing Attention data are different than those governing Data_a and Data_b. Attention data can be transmitted even though the LOCAL_CREDIT total in the slot block is zero. The purpose of Attention data is:

- to allow a control slot to preempt any data waiting to be transmitted at the local system

- to allow a control information to be transferred to the session partner even though the normal Data_a and Data_b paths are blocked due to lack of flow control credits

Each service class must define the maximum size of attention data slots. The slot block must reserve a buffer of this size dedicated to the delivery of Attention data. A semaphore is set by the slot layer when Attention data is delivered into this buffer and cleared when the user has process the data. If the semaphore is set when Attention data is delivered, the Attention data is discarded by the slot layer.

4.2.2 User/Slot Layer Interface

This interface allows users to advertise service, establish user (slot) sessions, and transmit and receive data. These three categories of service are referred to as directory services, session control services and data transfer services.

The slot layer itself formats slots for transmission and validates received slots before passing the data to the user. The slot layer is also responsible for flow control and periodic service advertisements.

4.2.2.1 Summary Of Functions

The slot layer offers the following hierarchy of functions to the user layer:

Table 4–5: User/Slot Layer Functions

Function offered:	Type of function:
H- start_service_class	directory service
S- poll_service_class	directory service
S- start_session (connect)	session service
H- new_session_poll	session service
H- accept_new_session	session service
H- reject_new_session	session service
SH- poll_session	session service
SH- queue_rcv_slot_buffer	data service
SH- poll_rcv_done	data service
SH- queue_attention_buffer	data service
SH- poll_attention_done	data service
SH- volunteer_xmt_data_a	data service

Table 4–5 (Cont.): User/Slot Layer Functions

Function offered:		Type of function:
SH-	volunteer_xmt_data_b	data service
SH-	volunteer_attention_data	data service
SH-	poll_xmt_done	data service
SH-	end_session (disconnect)	session service
H-	end_service_class	directory service

4.2.2.2 Description Of Functions

The functions offered to the user by the slot layer are:

- H-start_service_class(class,timer;status),
  H-end_service_class(class,reason;status),
  S-poll_service_class(class;status)

  These functions are used to advertise availability and to determine availability of particular service classes (such as interactive terminals) in the local area. Conceptually, these functions bracket all of the other services offered to the users by the slot layer. These are directory services, and are not an integral part of an implementation. These functions normally control the transmission and reception of multicast addresses.

  - class - service class (see appendicies)
  - timer (optional) - specified in seconds, determines frequency of multicast message transmission HOST_MULTICAST_TIMER (see DEFINED PARAMETERS AND RECOMMENDED OR REQUIRED DEFAULT VALUES).
  - status - one of: class enabled, class disabled.

- S-start_session(rem_srv_nm,loc_srv_nm,class,min_slot_size; handle,status)
  SH-end_session(handle,reason;status)

  These functions are used to establish a new session and to disestablish an existing session. These functions bracket the remaining functions offered to a user by the slot layer; data services cannot be invoked unless the user has successfully established a session.

  - address - Ethernet address of host

- rem_service_name - The name of the host service selected by the user. This field may not be null.
- loc_service_name - The name of the local service (username) or null.
- class - service class
- handle - used to reference the session locally
- slot_size - the minimum slot size queued by queue_rcv_slot_buffer for Data_a, Data_b and Attention slots.
- status - one of: request_active, request_rejected (see service class appendix), insufficient resources to complete request, no such session.
- reason - the reason can be an integer value, a byte counted ASCII string or both. This reason may be supplied to the users if that is appropriate to the implementation. The reason is conveyed to the remote half-session by the Stop message and/or a multicast message.

- H-new_session_poll(;handle,status),
  H-accept_new_session
  (hndl,rem_slt_nm,loc_slt_nm,min_slt_size,mode; status),
  H-reject_new_session(handle,reason;status)

  These functions are used to accept or reject a request to establish a session.

  - handle - used to reference the session locally
  - rem_slot_name - The slot_name received in the start slot or null.
  - loc_slot_name - The name of the local slot block (could be the host service name or some other name) or null.
  - slot_size - the minimum slot size queued by queue_rcv_slot_buffer for Data_a, Data_b and Attention slots.
  - status - one of: request active, request_rejected(see service class appendix), insufficient resources to complete request, no such session.
  - reason - the reason can be an integer value, a byte counted ASCII string or both.

- SH-poll_session(handle;status,quality)

  This function is used to determine the status of existing sessions.

  - handle - used to reference the session locally
  - status - Starting, Running or Halted (with reason code).
  - quality - the quality of the virtual circuit, one of: VC_ok, VC_suspect, transport disabled.

- SH-queue_rcv_slot_buffer(handle,buffer;status),
  SH-poll_rcv_done(handle;buffer,status)

  These functions allow slot buffers to be queued and received slot data to be delivered. The minimum length of this receive buffer is specified in the Start slot MINIMUM_ATTENTION_SLOT_SIZE and MINIMUM_DATA_SLOT_SIZE fields. Since slot credits are used for both Data_a and Data_b slots, these slot receive buffers must be the same minimum size.

  - handle - a reference to a local session
  - buffer - the starting address and length of a buffer.
  - status - one of: buffer queued, no slot data available, slot data available (Start, Stop, Data_a, Data_b or Reject).

- SH-queue_attention_buffer(handle,buffer;status),
  SH-poll_attention_done(handle;buffer,status)

  These functions allow the user to receive out of band data. Data delivered by this function is not sequenced relative to the data delivered via poll_rcv_done. If the data delivered by this function is not received faster than it is delivered, new data may be discarded by the slot layer.

  - handle - a reference to a local session
  - buffer - the starting address and length of a buffer.
  - status - one of: buffer queued, no data available, data available, data available and data missed.

- SH-volunteer_xmt_data_a(handle,buffer;status),
  SH-volunteer_xmt_data_b(handle,buffer;status),
  SH-volunteer_xmt_attention(handle,buffer;status),
  SH-poll_xmt_done(handle;buffer,status)

These functions allow data and attention slots to be transmitted to the session partner. Attention data is not flow controlled. Attention data (out of band) is not blocked by the normal data (in band).

- handle - a reference to a local session
- buffer - the starting address and length of a buffer.
- status - one of: buffer queued, data transmitted.

4.2.3 Slot/Virtual Circuit Layer Interface

This interface allows the slot layer to establish virtual circuits and to transmit and receive messages and multicast datagrams.

The virtual circuit layer maintains virtual circuits to one or more remote systems, transmits and receives messages, runs a timer, transmits and receives multicast datagrams and notifies the slot layer about changes in service.

4.2.3.1 Summary Of Functions

In summary, the virtual circuit layer offers the following hierarchy of functions to the slot layer:

Table 4-6: Slot/Virtual Layer Functions

Function offered:	
SH-	queue_rcv_datagram
SH-	poll_rcv_done
SH-	queue_transmit_datagram
SH-	poll_transmit_done
S-	VC_start
H-	accept_virtual_circuit
SH-	poll_virtual_circuit
SH-	poll_receive_message_done
SH-	queue_transmit_message
SH-	poll_transmit_message_acked
SH-	transmit_unacknowledged_queue
SH-	VC_stop

4.2.3.2 Description Of Functions

The specific functions offered to the slot layer by the virtual circuit layer are:

- S-Virtual_Circuit_start(node_name,max_sessions;handle,status),
  H-accept_virtual_circuit(handle;status),
  SH-Virtual_Circuit_stop(handle;status)

  These functions allow the slot layer to establish and disestablish virtual circuits. The slot layer must poll the virtual circuit layer to discover the state of virtual circuits and the state of the underlying data link itself. The host node must not dally in responding to a request to start a virtual circuit. If the host node wishes not to have any new circuits established, all service classes should be disabled and the appropriate status should be indicated in the multicast message transmitted by the host periodically. This multicast message should be transmitted at least once before host services are discontinued.

  - address - Ethernet address of destination system
  - node_name - The name of the target node. Used as the virtual circuit name CIRCUIT_NAME.

- max_sessions (optional) - the maximum number of session that will ever be active simultaneously. If supplied by the terminal server, a host implementation might avoid allocating unnecessarily large data structures.
- handle - handle used to reference the virtual circuit locally (a reference to the circuit block)
- status - one of: insufficient resources to complete request, no such circuit.

- H-poll_virtual_circuit(;handle,status,quality),
  SH-poll_virtual_circuit(handle;status,quality)

  These functions allow the existence of new sessions and the status of existing sessions to be determined.
  - handle - handle used to reference the virtual circuit locally
  - status - one of: Starting, Running or Halted (with reason).
  - quality - the virtual circuit quality, one of: VC_ok, VC_suspect, Ethernet data link disabled.

- SH-queue_receive_datagram(buffer;status),
  SH-poll_rcv_done(;handle,buffer,status)

  This function gives the virtual circuit layer datagram buffers which are in turn queued to the Ethernet data link. These datagram buffer are filled by multicast datagrams and by virtual circuit messages.
  - buffer - address and length of a buffer.
  - status - one of: buffer queued, no data available, multicast datagram available.

- H-queue_transmit_datagram(buffer;status),
  H-poll_transmit_done(;buffer,status)

  These functions queue datagrams to the Ethernet data link for transmission and poll for the transmit completion. These functions are used to transmit the multicast (or possibly other types) of datagrams.
  - buffer - address and length of a buffer.
  - status - one of: buffer queued, transmitter error.

- SH-queue_transmit_message(handle,buffer;status), SH-poll_transmit_message_acked(handle;buffer;status)

  These functions allow the slot layer add messages to the virtual circuit layer unacknowledged message queue and to receive them back after they have been acknowledged.

  - handle - handle used to reference the virtual circuit locally
  - buffer - address and length of a message.
  - status - one of: message queued for transmission or message transmitted.

- SH-transmit_unacknowledged_queue(handle)

  This function caused all outstanding unacknowledged messages to be retransmitted.

  - handle - handle used to reference the virtual circuit locally

- SH-poll_receive_message_done(handle;buffer,status)

  This function allows the slot layer poll the transport layer for any received messages.

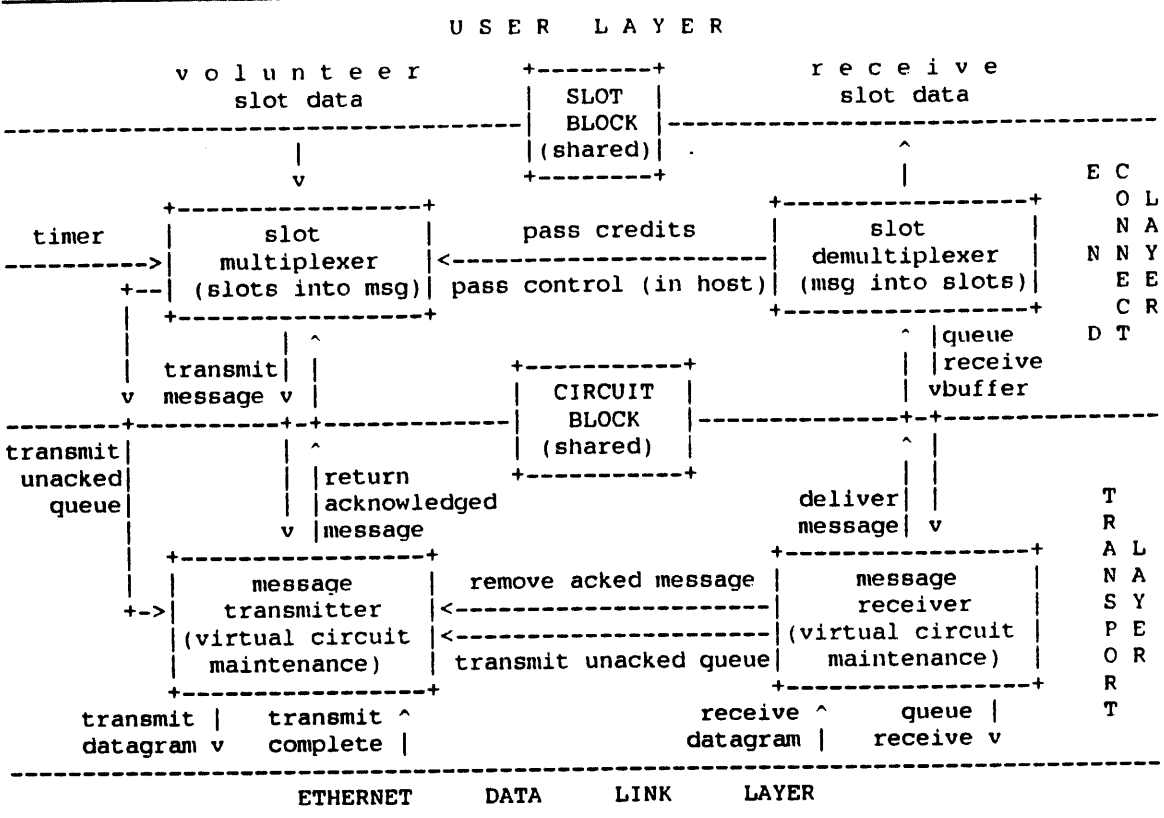
  - handle - handle used to reference the virtual circuit locally
  - buffer - address and length of a message.
  - status - one of: no data available, message available.

4.3 Axioms And Algorithms

This section details a set of algorithms that would produce the correct behavior at the user and Ethernet data link interfaces. While these algorithms would produce the desired result, any number of equivalent algorithms that produce an equivalent result at the same two interfaces would serve equally well.

The whole architecture can be characterized as two (host and terminal server) two-port black boxes. Compressing both the host and the terminal server into single diagram, one might visualize the internal structure of the LAT architecture as:

Figure 4-1: LAT Layers Interface



The following process abstractions are (somewhat arbitrarily) created within the slot layer to help present a detailed model of the internal flow of control and data within the layer:

- slot_demultiplexer - turns a message into one or more slots
- slot_multiplexer - turns one or more slots into messages
- session_starter - allocates a SLOT_BLOCK and initializes a half-session
- session_ender - closes a half-session and deallocates the SLOT_BLOCK
- administrator - advertises service (in host) and builds lists (in terminal server)

The following process abstractions are (again somewhat arbitrarily) created within the virtual circuit layer to help present a detailed model of the internal flow of control and data within the layer:

- circuit_starter (terminal server only) - starts new virtual circuits

- circuit_ender - stops an existing virtual circuit
- message_receiver - receives datagrams from the Ethernet data link, validates the received datagrams (turning each into a message) and passes the message on to the slot layer
- message_transmitter - receives messages from the slot layer, maintains a queue of unacknowledged messages, and transmits datagrams on the Ethernet data link.

4.3.1 Virtual Circuit Layer

The algorithms described in this section correspond to the state table actions, not the state table events.

4.3.1.1 Circuit Starter (Terminal Server Only)

When a request is received to start a new virtual circuit (via the VC_start function), the terminal server:

- allocate a circuit block (see state diagram) and buffers. Allocate NBR_DL_BUFS+1 receive message buffers and one transmit message buffer where NBR_DL_BUFS is the value sent in the Start message.
- queue the receive message buffer(s) to the Ethernet data link via the queue_rcv_datagram function
- store the transmit message buffer in the circuit block
- Translate the CIRCUIT_NAME into a 48-bit Ethernet address and load this address into the REM_ADDRESS field of the Circuit Block.
- generate a Start message (using the transmit message buffer) and queue it to the Message Transmitter via the queue_transmit_message function

4.3.1.2 Data_Volunteered

If Slot Data is volunteered:

- Set DRF in the coresponding slot block.
- If a credit exists to transmit the Slot Data, set the corresponding Circuit Block DWF.

4.3.1.3 Credits Returned

If:

- the current credit total is zero and
- a credit is being added to the Slot Block and
- the DRF is set in the Slot Block

then set the corresponding Circuit Block DWF.

4.3.1.4 Circuit Ender

When a request is received to stop an existing virtual circuit (via the VC_stop function), the Circuit Ender generates a Stop message and queues it to the Message Transmitter via the queue_transmit_message function and indicates in the circuit block that the virtual circuit is in the Halted state.

In the host, if service is being terminated, the Circuit Ender should also send at least one multicast datagram to indicate that service has been ended.

4.3.1.5 Message Receiver

The message receiver validates received messages. The Ethernet data link verifies the:

- Ethernet destination address of the frame matches that of the local system
- LAT protocol type (depending on implementation)

After these fields are checked, the message type is determined from the LAT header MESSAGE_TYPE field. The received message is then mapped as one of the message types (see STATE DIAGRAMS section). If any of the actions described fails, the message is discarded without perturbing the existing circuit state. In any case, the receive buffer is always queued back to the Ethernet data link quickly with respect to SERVER_CIRCUIT_TIMER.

- Start message (slave) -
  - discard message if the SRC_CIR_ID field of message is zero.
  - Match to an existing circuit block or, if no circuit block exists, allocate a circuit block and buffers. If insufficient resources exist to accomplish this, a best effort attempt is made to send a Stop message. Normally one receive message buffer is allocated. If the NBR_DL_BUFS field in the

Start message response will be non-zero, that many additional receive buffers are allocated. Two transmit message buffers plus (optionally) the value in the NBR_DL_BUFS in the received Start message are allocated. (One transmit message buffer is consumed by a message containing data with the RRF flag clear since it will not be acknowledged; so a second buffer is required which, if it is the last, will always have RRF set to force a terminal server response).

- queue the receive message(s) to the Ethernet data link via the queue_rcv_datagram function
- store the available transmit message buffers in the circuit block
- copy the SRC_CIR_ID field from the received message to the REM_CIR_ID field of the circuit block, copy the SRC_ADDRESS from the received message into the Circuit Block and copy the MASTER_NODE_NAME from the Start message in to the Circuit Block.
- generate a Start message (normally) or generate a Stop message (with a reason specified) and queue it to the Message Transmitter via the queue_transmit_message function

- Start message (terminal server) -
  - If the SRC_CIR_ID is non-zero, match to an existing circuit block. If no circuit block is referenced (invalid message) discard the message and send a Stop message addressed too SRC_CIR_ID. If the SRC_CIR_ID is zero it is an illegal message. If a valid Start message is received, SRC_CIR_ID of the received message should be copied to the REM_CIR_ID field of the referenced circuit block.
  - if the NBR_DL_BUFS field in the received Start message is non-zero, optionally allocate that many additional transmit buffers and store them in the circuit block.
- Run_msg_rcv (Host and terminal server) -
  - If the MSG_SEQ_NBR is not equal to ACK+1 (modulo 256) in the circuit block, set the NBR_SLOTS in the message header to zero. In the terminal server only, if the MSG_SEQ_NBR is not equal to ACK+1, copy the SOURCE_ADDRESS of the received message into the CIRCUIT_BLOCK REM_ADDRESS field - this will allow dynamic path failover if a host node has access to two different Ethernet ports.
  - In terminal server, if RRF is set, set DWF.

- request the Message Transmitter to return all messages acknowledged by the value MSG_ACK_NBR received in the message. This algorithm must be done modulo 256.
- In the host node, stop the HOST_RETRANSMIT_TIMER if all messages are acknowledged.
- If the MSG_SEQ_NBR is equal to ACK+1 (modulo 256) in the circuit block then increment ACK in the circuit block.
- pass the message to Slot demultiplexer

- Stop_rcv (or Reject_rcv) -
  - notify slot demultiplexer of circuit state transition
  - indicate state of circuit block is Halted
  - requeue the datagram buffer to the Ethernet data link

  Any time the circuit goes into the Halted state, the circuit block can be deallocated along with its associated resources.

A special use of the DST_CIR_ID field occurs whenever the host node and terminal server are out of synchronization due to crashes, prolonged communication link failures or nonsequential message delivery. In these cases, the remote system might ignore the message (possibly a delayed Stop message) in order to preserve the currently Running virtual circuit. In order to close this half established (half-open session), the SRC_CIR_ID field is copied from the Running message into the REM_CIR_ID field of the circuit block and a Stop message is generated. This will cause the remote system to reinitialize.

Any fields not defined by the LAT architecture (labeled UNPREDICTABLE in the message formats) are ignored.

4.3.1.6 Message Transmitter

The terminal server message transmitter normally maintains an unacknowledged transmit queue of one entry, while the host node normally maintains a queue of one or two entries. If extra data link buffers are allocated, these queues can be longer.

The Message transmitter gets three types of requests:

- transmit unacknowledged transmit queue - this causes the Message Transmitter to start transmitting the head of the queue, wait for the transmit complete

event, and transmit the next message until the entire queue has been emptied. If this was in progress when the request to retransmit the queue is made, the request is ignored.

Before a message is retransmitted, the MSG_ACK_NBR field in the message header is copied from the circuit block field ACK.

The Retransmit limiter is a part of the message transmitter. Every time the head of the unacknowledged transmit queue is transmitted, the retransmit counter is incremented (either HOST_RETRANSMIT_COUNTER or SERVER_RETANSMIT_COUNTER). Every time the head of the unacknowledged transmit queue changes, Retransmit_Counter is zeroed. If the retransmit counter reaches LAT_MESSAGE_RETRANSMIT_LIMIT, the Resend_limit event occurs. This event (Resend_limit) should cause users to be notified of the unacceptable virtual circuit quality.

- queue/dequeue transmit message - this adds and deletes entries from the unacknowledged transmit queue. As each new message is added to the queue, via the queue_transmit_message function, the message the header is created by:
  - copying the REM_CIR_ID from the circuit block to the message field DST_CIR_ID
  - copying the LOC_CIR_ID from the circuit block to the message field SRC_CIR_ID
  - (host only) the RRF flag is:
    - set if:
      1. the Circuit Block DWF is set or
      2. this is the last transmit message buffer
      3. credit consuming slots were sent in last message
      4. (optionally) if new data is expected via the volunteer functions - this may cause better behavior under load by preventing unnecessary "unsolicited" messages from being sent by the host. This state could be anticipated if data were delivered but not echoed for instance.
    - cleared otherwise.
  - (host only) copying the RRF flag state into the message header from the circuit block

- copying the NXMT circuit block field into the message header MSG_SEQ_NBR and incrementing NXMT

- transmit datagram - this function transmits the buffer and returns the transmit complete event, along with the buffer, to the requestor.

Retransmission of the unacknowledged transmit queue (the first of the three types of request described above) in the host node occurs at the rate HOST_RETRANSMIT_TIMER seconds. This causes messages to be retransmitted about every one or two seconds.

In the terminal server, this would be an unsatisfactory arrangement due to the rate at which retransmissions would occur (the value used by SERVER_CIRCUIT_TIMER is typically 80 milliseconds). Because the most likely reason a message is being retransmitted is that the host node has not had a chance to process the received message, the terminal server retransmission of messages must occur at dramatically reduced rates. One acceptable policy is to retransmit at SERVER_RETRANSMIT_TIMER intervals after the original message was sent.

NOTE

In an actual implementation, the host node may be overloaded and unable to respond to received buffers. The retransmit policy is based on the assumption that the host node has not responded because it has not processed the buffer. This policy assures that the host node is not swamped with duplicate buffers during heavy host loading.

4.3.1.7 Circuit Timer Policy

The circuit timers, in both the terminal server and the host node, should be reset both when the message is queued for transmission and when the transmit completes. This policy prevents multiple terminal servers from synchronizing by utilizing the Ethernet backoff algorithm.

4.3.1.8 Buffering

The LAT architecture assumes that any receive buffers assigned to the Ethernet data link cannot be preempted by other architectures that might share the data link. Failure to adhere to this policy may cause LAT to be unable to deliver data in a timely fashion.

In the case of a host implementation, the initial processing of received data link buffers should occur at high priority. Received messages that are out of sequence or received start messages that cause the current total to exceed the value LAT_

MAX_SERVERS, must be rejected immediately and requeued to the Ethernet data link to allow new data messages to be stored until they can be processed. Failure to adhere to this policy can cause long delays since host buffers can be filled by duplicates causing non-duplicates to not be delivered. If the retransmission policy is to wait one second, this failure mode will cause one second delays as perceived by the user.

4.3.2 Slot Layer

The algorithms described in this section correspond to the state table actions, not the state table events.

4.3.2.1 Host System Management

When the start of service is announced in the host (start_service_class function), a transmit datagram must be allocated and reserved for the purpose of transmitting the multicast datagram periodically. In addition, LAT_MAX_SERVERS receive datagram buffers must be allocated and queued via the queue_receive_datagram function. The value LAT_MAX_SERVERS is equal to the number of terminal servers that the host node wishes to allow to startup simultaneously.

These same resources must be recovered when the end of service is announced.

- Start multicast transmit and start multicast timer for retransmission.
- Stop the HOST_RETRANSMIT_TIMER (the timer could not be active since no virtual circuits are active).

4.3.2.2 Terminal Server System Management

When service is started in the terminal server:

- at least one buffer should be queued to the data link to receive multicast addresses.
- start the server circuit timer in anticipation of virtual circuits being activated (it should already be running).

Policies might reasonably be modified or extended by each different service class.

4.3.2.3 Session Starter (Terminal Server)

This process allocates and initializes a slot block upon receiving a call from the start_session function.

One important responsibility of the session starter of the terminal server is translating the REM_SERVICE_NAME supplied in the start_session function into a node_name to be passed in the VC_START function when a new virtual circuit must be established.

4.3.2.4 Session Starter (Host)

A host service implementation can choose between two models. In one model, received Start slots cause the user to receive a request to start a new session. The user can then either accept or reject the session. The user should not procrastinate in making this decision.

The second model does not give the host service this choice, but instead accepts or rejects the session without notifying the service. Later the host service may stop the session with a reason.

If a DST_SLOT_NAME is specified in the start slot, the host session can be bound to a specific host service (host port such a specific controller or unit) associated with the slot name or can be bound to a more abstract service, such as electronic mail.

4.3.2.5 Slot Demultiplexer

This process receives its input and control from the poll_rcv_done function. Each such message received has been validated on the virtual circuit by the message receiver.

Messages contain zero or more slots. A slot can be a Start, Data_a, Data_B, Attention, Reject, or Stop slot.

Slots are validated by using the received slot's DST_SLOT_ID field to reference a slot block. The referenced slot block's LOC_SLOT_ID field must equal the received slot's DST_SLOT_ID field. Additionally, the received slot's SRC_SLOT_ID field must equal the slot block's REM_SLOT_ID field. The slot type field must be consistent with the state block receiving the slot. (See the section on slot mapping onto state diagram.)

If the terminal server receives a start slot, the slot's SRC_SLOT_ID field is copied into the referenced slot block's REM_SLOT_ID field.

In the host, the slot_demultiplexer creates and initializes a slot block if a start slot is received and passes the slot block to the appropriate class of service via the new_session_poll function. The slot block is a data structure shared by all slot layer processes and is accessed by the user processes. (see slot state variables section).

The host may use the DST_SLOT_NAME supplied in the Start slot to bind the session to a particular service access point.

If any credits are received in a slot, the credit field value is added into the LOCAL_CREDIT field of the referenced slot block to create a new total.

Data_a and Data_b slots are delivered via the poll_rcv_done; Attention slots are delivered via poll_attention_done; Start and Stop slots are delivered via the poll_session function.

A Stop slot causes the slot block to be deallocated and the event is delivered to the user via the poll_session function.

In the host, after the slot demultiplexer has finished processing the received message, the message is requeued to the virtual circuit layer and control is passed to the slot multiplexer.

4.3.2.6 Slot Multiplexer

In the host, this process receives control soon after the slot demultiplexer executes. This transfer of control can be immediate or can be delayed in an effort to return more data in the response. This transfer of control between the slot demultiplexer and the slot multiplexer in the host does not have to preserve the "serial and atomic" requirement stated in the layer interface introduction. This delay should never exceed about 1/2 the SERVER_CIRCUIT_TIMER; the response generated must be received by the terminal server before SERVER_CIRCUIT_TIMER expires a second time.

In the terminal server, this transfer of control is timer based and cannot be less than SERVER_CIRCUIT_TIMER milliseconds.

When reading the algorithms, keep in mind that "slot data" can be Data_a, Data_b, Attention data or REMOTE_CREDITS waiting to be transferred.

Before accepting any data from users, the process verifies that at least one transmit message buffer is available in the circuit block. If none is available, then execute the transmit_unacknowledged_transmit_queue function. The following algorithm is executed by the terminal server whenever control is received

from the timer event and by the host whenever control is received from the slot_demultiplexer. In the host, transfer of control is also received from the volunteer functions if the circuit block RRF flag is clear (the Send_data event):

- In the host, if the DWF is clear:
  - dequeue a transmit message buffer from the circuit block XMT_BUFFER_FREEQ (if it is empty, execute the transmit_unacknowledged_queue function and exit)
  - generate a new message header
  - execute queue_transmit_message
  - execute transmit_unacknowledged_queue
- In the terminal server, if the DWF is clear, exit.
- if the DWF is set:
  1. dequeue a transmit message buffer from the circuit block XMT_BUFFER_FREEQ (if it is empty, go to step 8 - EXIT)
  2. generate a new message header
  3. UNTIL the message buffer is filled or UNTIL all slots have DRF clear or UNTIL all slots with DRF set have no LOCAL_CREDITS left: Find the next slot block with DRF set in a round-robin fashion and:
     - if Attention data is volunteered, format attention slot in message buffer, clear DRF if no slot data is left. Exit back to UNTIL loop.
     - if Data_a or Data_b is signaled, decrement LOCAL_CREDITS (if none available, go to next step), format Data_a or Data_b slot header in message buffer, copy REMOTE_CREDITS field from slot block into slot header and zero and REMOTE_CREDITS field, copy data into slot and clear DRF if no slot data is left. Exit back to UNTIL loop.
     - if REMOTE_CREDITS is non-zero (because LOCAL_CREDITS is zero in previous step or because no Data_a or Data_b has been volunteered), format Data_a slot header in message buffer, copy REMOTE_CREDITS field from the slot block into slot header and zero the slot block REMOTE_CREDITS field, and clear DRF if no slot data is left. Exit back to UNTIL loop.
  4. queue the buffer via queue_transmit_message

5. if all slot block DRF flags are clear, clear DWF.

6. In the host, if control was received via a volunteer function, the HOST_RETRANSMIT_TIMER is started. (Whenever this timer expires, all unacknowledged messages are retransmitted.)

7. If more slot data is available, go to step 1.

8. Exit - execute the transmit_unacknowledged_queue function.

4.3.2.7 Session Ender

The terminal server cannot stop a session while it is in the Starting state. This is because the handle on the remote slot block is not known until a response to the start slot is received. Thus the special state Abort_start is entered upon receiving a disconnect_req. A slot block in this state cannot be used to start a new session.

Otherwise, either the host service or the terminal server user can issue an end_session (disconnect) function call.

4.3.2.8 Flow Control

There are two levels of flow control: One in the slot layer and one in the virtual circuit layer.

4.3.2.9 Slot Flow Control

The session user that owns a flow control credit (credit is held on user's behalf by the slot layer), is guaranteed that the session partner will be able to receive (and buffer) at least one Data_a or Data_b slot. In an implementation, slot data is copied from a receive message buffer in the virtual circuit layer into slot buffers supplied by the users. This is the model described in this document.

These flow control credits are consumed by Data_a and Data_b slots if, and only if, the SLOT_BYTE_COUNT field is non-zero.

As a CPU performance optimization, this can be implemented in different way. CPU usage can be traded for memory usage. The users can supply data link sized slot buffers (LAT_MIN_RCV_DATAGRAM_SIZE), and the entire buffer can be passed to the user without copying data. This method requires that an occupancy count be maintained for each buffer since buffers can be occupied by one or more slots destined for different users. In this way received slot data does not have to be copied. However prodigious amounts of memory is consumed. For instance, to extend two slot credits for each of 8 users, 2x8x1518 (24,288) bytes of buffering is used instead of 2x8x255 (4080) bytes of buffering.

References are made to "slot" flow control throughout the document. The sections SLOT MULTIPLEXER and SLOT DEMULTIPLEXER in the AXIOMS and ALGORITHMS section define how slot flow control is applied to a running slot session.

4.3.2.10 Message Buffer Flow Control

In the virtual circuit layer, when a new circuit is to be established, a fixed number of datagram buffers is allocated before the Start message is sent. These buffers are queued as receive buffers to the Ethernet data link layer. The number of buffers queued as receive buffers minus one is then transmitted in the Start message NBR_DL_BUFS field.

In the terminal server, the number of transmit buffers allocated should be equal to the value NBR_DL_BUFS received in the Start message plus one.

In the host, the number of transmit buffers allocated should be equal to the value NBR_DL_BUFS received in the Start message plus two. This extra buffer is used to sent the one possible "unsolicited" message when the virtual circuit is balanced (RRF clear).

In general, this document does not directly refer to flow control at the virtual circuit level (references to the term "flow control" are normally references to slot flow control). Instead, references are made to the availability of data link buffers (XMT_BUFFER_FREEQ in the circuit block). Availability of a data link message buffer corresponds to the availability of a credit to transmit a message buffer since these buffers remain on the unacknowledged transmit queue until they are acknowledged by the receiving process. This acknowledgment guarantees that the remote system has emptied and requeued the data link buffer to receive a new message. See the AXIOMS and ALGORITHMS section titled MESSAGE TRANSMITTER and MESSAGE RECEIVER.

4.3.2.11 Protocol Versions And ECO Control

Multicast messages for each service class specify LOW_PRTCL_VER, HIGH_PRTCL_VER, CUR_PRTCL_VER, CUR_PRTCL_ECO.

The start message specifies PRTCL_VER and PRTCL_ECO.

The HIGH_PRTCL_VER specifies the highest (most recent) protocol version that the system supports.

The LOW_PRTCL_VER specifies the lowest (oldest) protocol version that the system supports.

The CUR_PRTCL_VER specifies the protocol version of the message. In the case of the Start message, it also guarantees that all other messages will also be of the same protocol version as the Start message.

The PRTCL_ECO specifies the Engineering Change Order level of the message. Again, in the case of the Start message, it also guarantees that all Run and Stop messages will also be of the same ECO level as the Start message. ECOs are made to a protocol version only if the change will not adversly affect the unchanged systems already in the field. This means that existing systems should continue to perform implemented functions, but from the point of view of the system implementing higher ECO level, not all of its functionality will be supported by the lower level system.

For any given protocol version, ECOs are backward compatible. If a change will make systems incompatible in the field, then a new (higher) protocol version number must be allocated.

4.3.3 Other Processes

4.3.3.1 Keep-Alive Process

The keep-alive process is relevant to the terminal server only - the host does not implement a keep-alive process. The purpose of the keep-alive process is to notify the users of an idle virtual circuit that the circuit is suspected to be inoperable. This is is accomplished by causing data to be transmitted at least every LAT_KEEP_ALIVE_TIMER seconds. The keep-alive process accomplishes this by simply guaranteeing that the data waiting flag (DWF) is set at least every LAT_KEEP_ALIVE_TIMER seconds.

Setting DWF causes a sequenced message to be sent. If the message repeatedly fails to be acknowledged, the LAT_MESSAGE_RETRANSMIT_LIMIT will be reached, and the users notified of the unacceptable circuit quality.

4.3.3.2 Progress Process

In theory, the virtual circuits described in this document cannot "deadlock". However, cosmic radiation, UNIBUSes and other equally defenseless culprits are often blamed for events that "cannot" happen.

As insurance against such unlikely events, a terminal server can implement a progress process. After the LAT_MESSAGE_RETRANSMIT_LIMIT is reached, an implementation may choose to continue sending messages every LAT_KEEP_ALIVE_TIMER seconds. If the value of LAT_MESSAGE_RETRANSMIT_LIMIT should reach a ridiculous value, such as 500 messages, or if more than an hour of real time has elapsed, the circuit should be stopped by transmitting a stop message with an appropriate reason.

A host implementation must run an additional timer when the RRF flag is clear in the circuit block. If a message is not received within a reasonable time (as little as 2 or 3 times the LAT_KEEP_ALIVE timer seconds or as long as a few days), the host may wish to generate a Stop message to stop the circuit. If host implementation lacked this timer, it would not discover that a terminal server had crashed if the crash occurs while the host RRF flag was clear. The hazard is that host resources are dedicated to the virtual circuit until a user from the same terminal server again requests service from the host.

4.4 Message Formats

Bits are transmitted onto the Ethernet low order bit first. When fields are concatenated, the right hand field is transmitted first. Numeric fields more that 8-bits long are transmitted least significant byte first.

Fields are represented as bit streams, right to left. All fields are an integer multiple of eight bits. The symbol " = " is used to indicate fields of varying or indeterminate length.

A message transmitted from a master to a slave (from a terminal server to a host node) always has the MASTER bit of the message type field set to 1. A message transmitted from a slave to a master always has the MASTER bit of the message type field set to 0. Notice that this makes it possible to implement both ends of the asymmetric LAT architecture simultaneously in a single system.

Legal values of the LAT messages are restricted by the LAT architecture to be in the range of 576 through 1518 bytes. Node may receive an unsolicited LAT message (Start) of maximum LAT message length and is responsible for allocating enough buffers to accomodate this message. Sizes of the solicited messages are specified during the virtual circuit establishment process through the LAT_MIN_RCV_DATAGR_SIZE value in Start messages. Once a circuit is established, all messages exchanged between those nodes are limited by the specified values. Note that specified sizes can be different on both sides of the connection. Nodes can also receive unsolicited Command and Solicit information messages. The size of the message that can be processed by the sending node is specified in the DATA_LINK_RCV_FRAME_SIZE value.

A LAT message is defined as only those fields specified in this specification; datalink fields are not included. I.e. the actual length of the LAT message that can be sent to the transmitting node equals the value given in those fields minus 18. (18 is the amount of overhead required by the standard Ethernet datagram. The Ethernet datagram is no longer part of the LAT specification, but compatibility requires addition of this value.)

Following rules define the value of the slot count byte field in the slot header:

- value of the slot count byte must include all bytes present in the slot;
- all fields defined by the architecture must be present in the slot;
- parameter code 0, indicating the end of the parameters list, must be present in the slot.

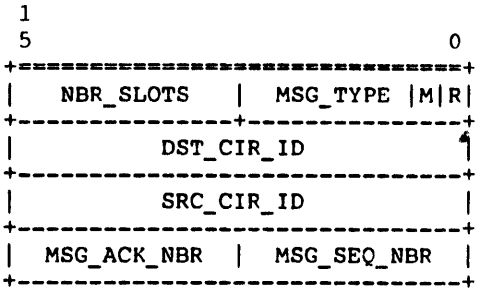
Exceptions to these rules are noted in the relevant sections.

When an error reason value of zero is specified, no reason has been supplied.

4.4.1 Virtual Circuit Message Header

All messages have the same header format:

Figure 4–2: Message Header Format



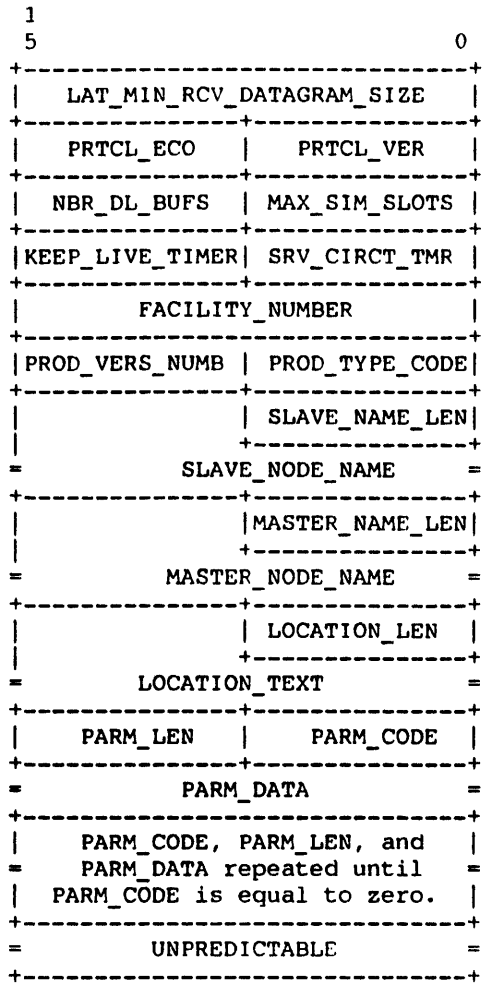
- R (1 bit) - RRF flag. This flag is clear for all message types except for Run messages transmitted from the host node to the terminal server which require responses. This flag is never set in any message transmitted by the terminal server.
- M (1 bit) - MASTER flag. This flag is set in all messages sent by the terminal server to the host node. Messages sent by the host node to the terminal server always clear this flag.
- MSG_TYPE (6 bits) - the message type field:
  - Start message have this field set to 1.
  - Run message have this field set to 0.
  - Stop message have this field set to 2.
- NBR_SLOTS - number of slots in the message

- DST_CIR_ID - one of the virtual circuit identifications
- SRC_CIR_ID - one of the virtual circuit identifications
- MSG_SEQ_NBR - message sequence number (modulo 256)
- MSG_ACK_NBR - message acknowledgment number (modulo 256)

4.4.1.1 Start Message Format

Start message headers have MSG_TYPE fixed at 1, the NBR_SLOTS equal to zero. Additionally, start messages transmitted by the terminal server must specify the DST_CIR_ID as zero and the SRC_CIR_ID as non-zero. Start messages transmitted by the host node must specify these same two fields as non-zero.

Figure 4-3: Start Message Format



- LAT_MIN_RCV_DATAGRAM_SIZE (2 bytes) - an implementation must specify the maximum LAT message size it is capable of processing. Actual length of a LAT message is LAT_MIN_RCV_DATAGRAM_SIZE-18.

- PRTCL_VER (1 byte) - The protocol version of this message and of all messages transmitted during this session (current version is 5).

- PRTCL_ECO (1 byte) - The protocol version ECO (Engineering Change Order) of this message and of all messages transmitted during this session (current ECO is 1).

- MAX_SIM_SLOTS (1 byte) - maximum number of simultaneous sessions that can be opened on this virtual circuit. Value is suggested by the terminal server. Value supplied by the host must be used as the maximum by the terminal server.

- NBR_DL_BUFS (1 byte) - number of extra data link buffers queued. This corresponds to the number of additional messages (beyond the normal one message) that can be generated by the slot multiplexer on the system receiving this start message.

- SERVER_CIRCUIT_TIMER (1 byte unsigned) - Circuit timer in 10 millisecond intervals. Specified by terminal server. This field is ignored when received from the host. A value of zero in this field is illegal.

- KEEP_ALIVE_TIMER (1 byte unsigned) - Value specified in seconds by terminal server. This field is ignored when received from the host by the terminal server. A value of zero indicates that no keep-alive message will be sent.

- FACILITY_NUMBER (2 bytes) - Value specified by the server and host. This value is not restricted. It is intended to allow terminal servers and hosts to be uniquely numbered within a local area. A privileged user should supply this value to the implementation.

- PROD_TYPE_CODE (1 bytes unsigned) - The product type codes are assigned by Digital Equipment Corporation.

- PROD_VERS_NUMB (1 byte unsigned) - Product version number.

- SLAVE_NAME_LEN (1 byte unsigned) - The byte count of the next field. A value of zero in this field is illegal.

- SLAVE_NODE_NAME (SLAVE_NAME_LEN bytes) - Name of the slave node of the connection.

- MASTER_NAME_LEN (1 byte unsigned) - The byte count of next field. A value of zero in this field is illegal.

- MASTER_NODE_NAME (MASTER_NAME_LEN bytes) - Name of the master node of the connection.

- LOCATION_LEN (1 byte unsigned) - Byte count of LOCATION_TEXT field. This field may be zero.

- LOCATION_TEXT (LOCATION_LEN bytes) - The text within this field should describe the physical location of the system that transmits this message.

- PARM_CODE (1 byte) - A parameter code. No parameter codes are currently defined. The value zero indicates the end of the list (which means the following fields are unpredictable). A non-zero value in this field indicates the next two fields are valid. Parameter codes 0 through 127 are reserved for use by Digital Equipment Corporation, while parameter codes 128 through 255 are reserved for users.

- PARM_LEN (1 unsigned byte) - the length of the following field in bytes.

- PARM_DATA (PARM_LEN bytes) - the format of this field is defined by the associated PARM_CODE.

4.4.1.2 Run Message Format

Run messages have MSG_TYPE set to 0. If the NBR_SLOTS (number of slots in the message) is zero, then the message header is the entire message. NBR_SLOTS is equal to the number of slots within the message. The DST_CIR_ID and SRC_CIR_ID must always be non-zero in Run messages.

Each slot is aligned on word (16-bit) boundaries. The first slot is contiguous to the message header. The second slot is contiguous to the first if the slot's total length is even. If a slot's total length is odd, then one byte of UNPREDICTABLE data is used as a pad byte between the slot to force the following slot to a word boundary.

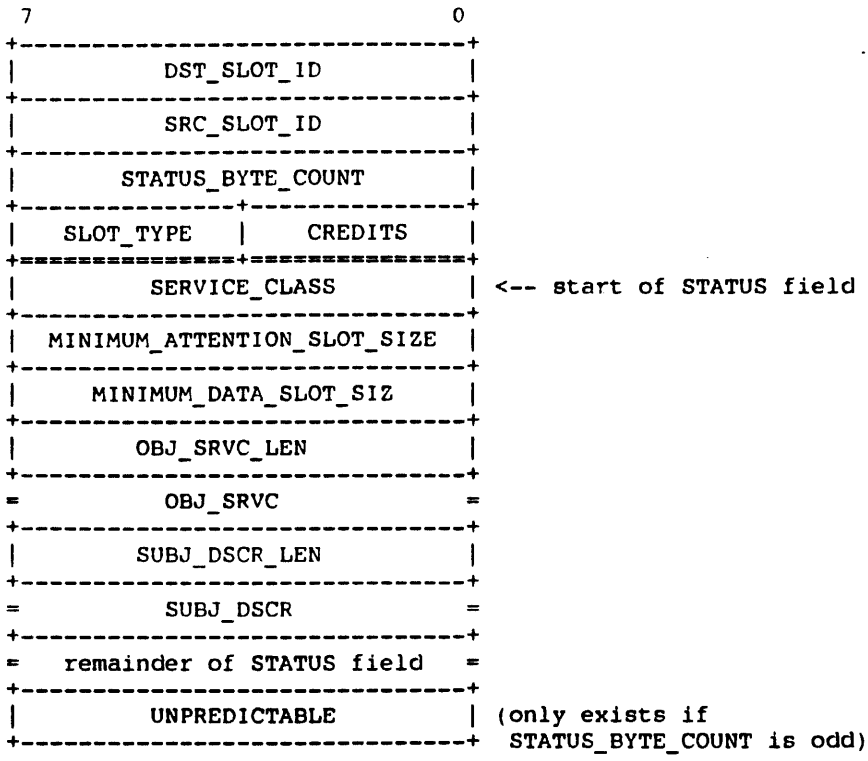
Run messages can contain Start, Data_a, Data_b, Attention, Reject and/or Stop slots.

Note that the slot type assignment are done to assist an implementation in detecting a Data_a slot. Specifically Data_a slots are assigned the value zero while all other slots (and all future slot type assignments) are assigned a four bit value with the left-most bit set. Thus Data_a slots are easily recognized since the byte value is always zero or positive and credits are conveyed by Data_a slots as a byte value.

4.4.1.3 Start Slot

If a start slot is received (see slot state tables), the format of the slot is:

Figure 4-4: Start Slot Format

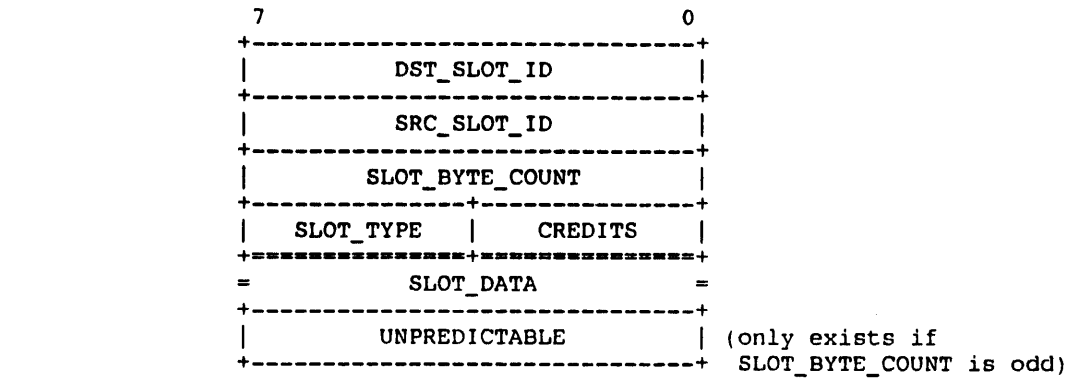


- DST_SLOT_ID - a reference to a slot block
- SRC_SLOT_ID - a reference to a slot block
- STATUS_BYTE_COUNT - an unsigned integer count of the length of the STATUS field.
- CREDITS (4 bits) - a 4-bit integer equal to the number of credits being transferred.
- SLOT_TYPE (4 bits) - the value 9 (1001).
- SERVICE_CLASS - see appendices
- MINIMUM_ATTENTION_SLOT_SIZE (1 byte) - The minimum slot size queued to receive Attention slot data (not including the slot header). The system receiving this message must limit transmitted Attention slots to this size. A value of zero indicates Attention slots are not supported.

- MINIMUM_DATA_SLOT_SIZE (1 byte) - The minimum slot size queued to receive Data_a and Data_b slots (not including the slot header). The system receiving this message must limit transmitted Data_a and Data_b slots to this size.
- OBJ_SRVC_LEN (1 byte unsigned) - The byte count of the next field.
- OBJ_SRVC - The name of the destination service.
- SUBJ_DSCR_LEN (1 byte unsigned) - The byte count of the next field.
- SUBJ_DSCR - The description of the source service.
- STATUS - The remainder of the status field meanings are defined separately for each service class.

4.4.1.4 Data_a Slot

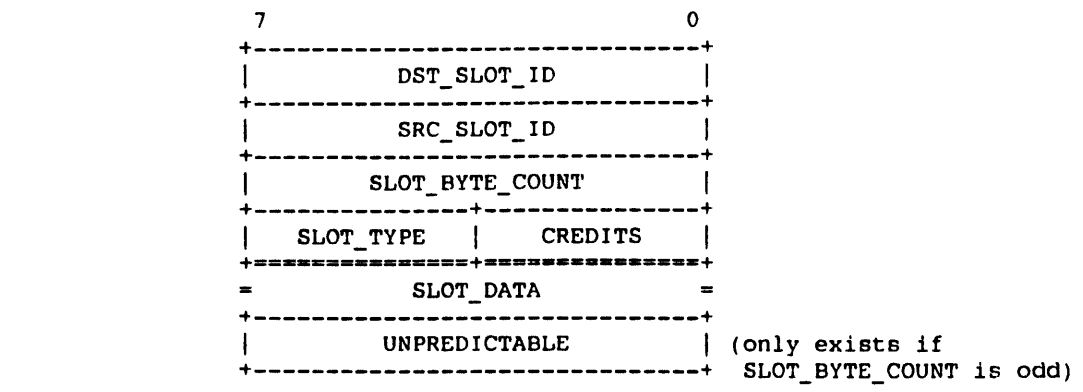
Figure 4-5: Data_a Slot Format



- DST_SLOT_ID - a reference to a slot block
- SRC_SLOT_ID - a reference to a slot block
- SLOT_BYTE_COUNT - an unsigned integer count of the length of the SLOT_DATA field.
- CREDITS (4 bits) - a 4-bit positive integer equal to the number of credits being transferred.
- SLOT_TYPE (4 bits) - the value 0.
- SLOT_DATA - SLOT_BYTE_COUNT bytes of slot data.

4.4.1.5 Data_b Slot

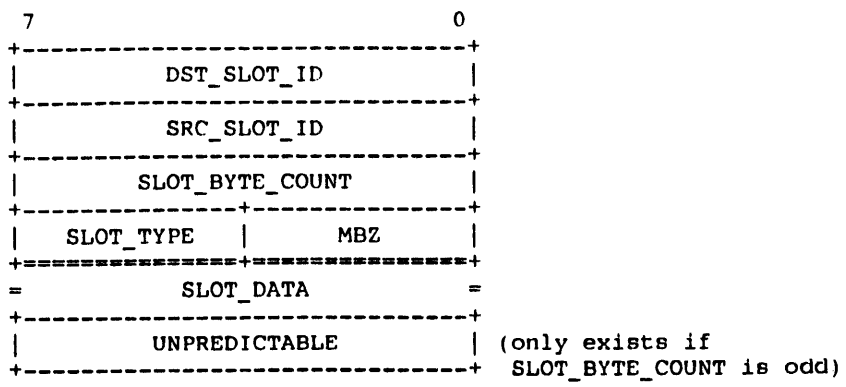
Figure 4-6: Data_b Slot Format



- DST_SLOT_ID - a reference to a slot block
- SRC_SLOT_ID - a reference to a slot block
- SLOT_BYTE_COUNT - an unsigned integer count of the length of the SLOT_DATA field.
- CREDITS (4 bits) - a 4-bit positive integer equal to the number of credits being transferred.
- SLOT_TYPE (4 bits) - the value 10. (1010)
- SLOT_DATA - SLOT_BYTE_COUNT bytes of slot data.

4.4.1.6 Attention Slot

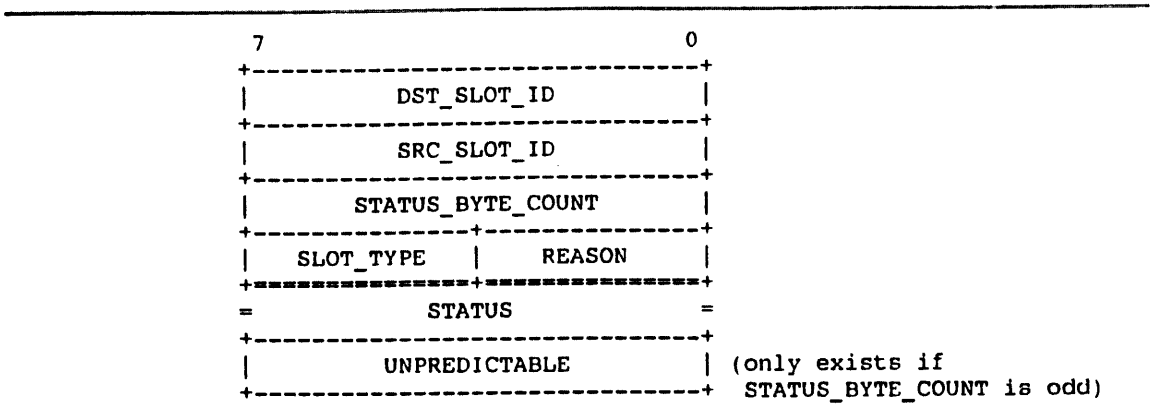
Figure 4–7: Attention Slot Format



- DST_SLOT_ID - a reference to a slot block
- SRC_SLOT_ID - a reference to a slot block
- SLOT_BYTE_COUNT - an unsigned integer count of the length of the SLOT_DATA field.
- MBZ (4 bits) - must be zero.
- SLOT_TYPE (4 bits) - the value 11. (1011)
- SLOT_DATA - SLOT_BYTE_COUNT bytes of slot data.

4.4.1.7 Reject Slot

Figure 4–8: Reject Slot Format

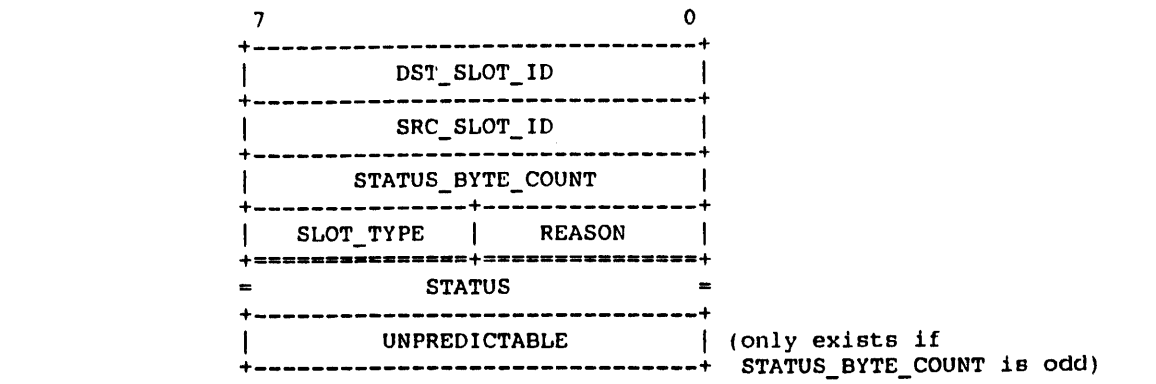


- DST_SLOT_ID - a reference to a slot block
- SRC_SLOT_ID - a reference to a slot block
- STATUS_BYTE_COUNT - an unsigned integer count of the length of the STATUS field.
- REASON - an unsigned 4-bit integer (see following section entitled "Slot reason codes").
- SLOT_TYPE - the value 12. (1100)
- STATUS - STATUS_BYTE_COUNT bytes of status. The status field meanings are defined separately for each service class.

This slot can only be transmitted from the host.

4.4.1.8 Stop Slot

Figure 4–9: Stop Slot Format



- DST_SLOT_ID - a reference to a slot block
- SRC_SLOT_ID - a reference to a slot block (must be zero)
- STATUS_BYTE_COUNT - an unsigned integer count of the length of the STATUS field.
- REASON - an unsigned 4-bit integer (see following section entitled "Slot reason codes").
- SLOT_TYPE - the value 13. (1101)
- STATUS - STATUS_BYTE_COUNT bytes of status. The status field meanings are defined separately for each service class.

4.4.1.9 Slot Reason Codes

Reason codes used in the Reject slot and Stop slot are defined below. These codes are also used by Status message (see section on the connection solicitation).

1. reason is unknown
2. user requested disconnect
3. system shutdown in progress
4. invalid slot received
5. invalid service class

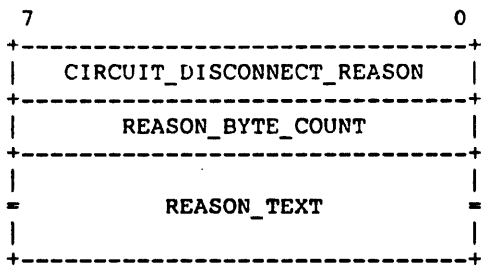
6. insufficient resources to satisfy request
7. service in use
8. no such service
9. service is disabled
10. service is not offered by the requested port
11. port name is unknown
12. invalid password
13. entry is not in the queue
14. immediate access rejected
15. access denied
16. corrupted solicit request

Note: all reason codes have been assigned.

4.4.1.10 Stop Message Format

Stop message headers have MSG_TYPE equal to 2. The SRC_CIR_ID field must always be zeroed in Stop messages.

Figure 4–10: Stop Message Format



- CIRCUIT_DISCONNECT_REASON (1 byte unsigned) - A zero value means no reason is given. The currently defined reasons are:
  1. reason is unknown
  2. No slots connected on virtual circuit.
  3. Illegal message or slot format received.

4. VC_halt from user.
5. No progress is being made.
6. Time limit expired.
7. LAT_MESSAGE_RETRANSMIT_LIMIT reached.
8. Insufficient resources to satisfy request.
9. SERVER_CIRCUIT_TIMER out of desired range.
10. Number of virtual circuits is exceeded.
11. (make up your own reasons, but please get them added to this document).

- REASON_BYTE_COUNT (1 byte) - Byte count of REASON_TEXT field. Normally specified as zero.
- REASON_TEXT (REASON_BYTE_COUNT bytes) - This field of ASCII characters contains the reason the stop message was sent.

5

Connection Solicitation

5.1 Architectural Model

This section describes how an application process running within a host (slave) environment can initiate a connection to application terminals that are connected to the terminal server (master).

The primary difference between interactive and application terminals is that interactive terminals require a mechanism for master-initiated connections to slaves, while application terminals require a mechanism for slave-initiated connections to masters. Since the LAT protocol requires that only a master can actually start a connection to a slave, different processes are required to allow a slave to formulate a request to a master to start a connection. During that process, a slave initiates a connection by issuing a solicitation request to a master, which actually starts a connection (the connection solicitation process). The connection solicitation process also allows "queued" services, where connection requests can be stored in the service's queue for future processing. The architecture of the connection solicitation process implemented by LAT architecture not only allows slaves to initiate connections to masters, but also allows masters to use the connection solicitation process to initiate connections to slaves, thereby providing queued services on slave nodes.

To implement the mechanism providing the connection solicitation process, two aspects of the LAT architecture are discussed in the following chapter and in the Appendix A: local area directory service and connection initiation. Local area directory service includes the advertising process, which allows advertising of services offered by the masters. The connection initiation process supports connection solicitation and queued services.

5.1.1 Service Sharing

The principle of "service sharing" is based on a mechanism of queues. Each service possesses "queued" or "non-queued" characteristics. Queues are accessed by subjects through connection requests qualified by "queued" or "non-queued" access methods. When a subject makes a request for usage of a service, the subject specifies the access method to be used. The object returns an acknowledgment, where it specifies the status of the request. A request for "queued" access may be rejected, or it may be accepted in the service's queue for future processing. If the service is busy with another user, and the service possesses characteristics that permit queuing, and the user requests "queued" access, the connection request is placed in a queue associated with the service. The subject node stores the connection request in its context area and the object node that receives the connection request creates a corresponding entry in its queue. The LAT architecture provides a method of correlating the request from the subject with an entry in the object's queue using two values: the request identifier and the queue entry identifier. The queue status and entry information are passed back to the subject requesting the connection. Subject node operations are available to the subject to request the status of the whole queue and individual entries, cancel the queued entry, etc.

Service characteristics, available to a user through the Advertising process and the specification of an access type supplied by the subject during the connection request, allow a user to organize usage of services.

- Service characteristics are:
  - queued
  - non-queued
  - disabled
- Access types used by subjects are:
  - queued
  - non-queued

Allowed service access-service characteristics combinations are:

- Queued or non-queued access to a disabled service:
  - Any type of access to a disabled service will result in the "service disabled" reject reason returned.

- Queued access to a non-queued service:

  — If service is available, then access is granted and connection is made. If service is not available, then "immediate access rejected" error is returned.

- Queued access to a queued service:

  — A service is available for use potentially by many users. For this reason, request arbitration in the form of request queuing is provided by an object. This scheme provides an element of fairness since requests from the users are queued to the service on a first come, first served basis.

  — As requests arrive from subject nodes, they are queued to the service. They are then dequeued on a first-come, first-served basis. When its request is dequeued, the subject has exclusive use of the service until either the user disconnects from the service (i.e., finishes using it) or the server disconnects from the user's node.

  — When the current session completes, the next request is dequeued and causes an attempt to establish a connection between the subject and the object. Subject and object nodes exchange Start messages (Start slots) establishing a session between the user and the service.

  — When a session is established, data transmission occurs by means of Data Slots after the Start Slots are exchanged.

  — The subject that requested the connection can always refuse to start the connection or cancel the queued request. Such a situation may occur if the node found another service available to satisfy the request after it sent the original request. Note that a mechanism is provided to the node that requested a connection to cancel a queued request.

- Non-queued access to a queued service:

  — Non-queued access to a queued service will be accepted only if the queue is empty. If there are other entries in a queue an "immediate access rejected" error will be returned by the subject. After the connection is started, a user has exclusive access to a service. All requests for queued access will be queued, and for non-queued access will be rejected. An active user retains exclusive access to the service until the connection to the service is broken.

- Non-queued access to a non-queued service:

  — A "service in use" error will be returned in a Status message if the service is being currently used. Otherwise, an attempt to establish a connection will be made by the solicited node.

The connection solicitation process allows a subject to queue a connection request to a service that possesses characteristics allowing queuing. A subject requests queued access to a service, and the queued service places the connection request in the queue for future processing. The connection solicitation process provides coordination between the subject and the object, allows correlation of the request (subject) and corresponding queue entry (object),and provides the subject with set of functions to handle the queue. Communication between a subject and an object that provides queuing of a connection request is implemented using two messages - the Command message and the Status message. For specific details on messages flow and object/subject state-tables during the solicitation process, see "Connection Initiation."

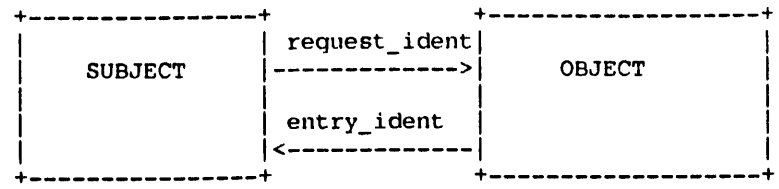
5.1.1.1 Queue Coordination

Coordination of a connection request between a subject and an object is implemented using two values that identify local data structures within each of the communicating nodes - the request_ident and the entry_ident. The request_ident and the entry_ident are unique in the node in which they where assigned and must not be 0. When system is booted, initially these identifiers must be chosen as random numbers. These identifiers may be reused by the node. The algorithm for assigning and deassigning of these identifiers is presented in the Appendix C.

To solicit a connection, the subject node assigns a request_ident (a handle on the local data structure) and sends it to the object node in a Command message. The object node that receives the solicitation request (and decides to queue it to a specific service) assigns an entry_ident to its local data structure that corresponds to the solicitation request, and sends the entry_ident back to the solicitor in the Status message. State-tables that fully describe the request-queue entry coordination process through message flow are defined in a later section.

The above identifiers allow communicating nodes to coordinate solicitation requests. Many requests can be queued to the service simultaneously, and each request is uniquely identified by the name of the node that queued the entry and a pair of the request-entry identifiers. Any request from the subject and any response from the object must supply in the corresponding messages the request_ident-entry_ident pair that uniquely defines the connection request. The request from the subject can be rejected by the object. The reason for the rejection is returned to the subject in the Status message (see LAT messages).

Figure 5–1: Queue Coordination

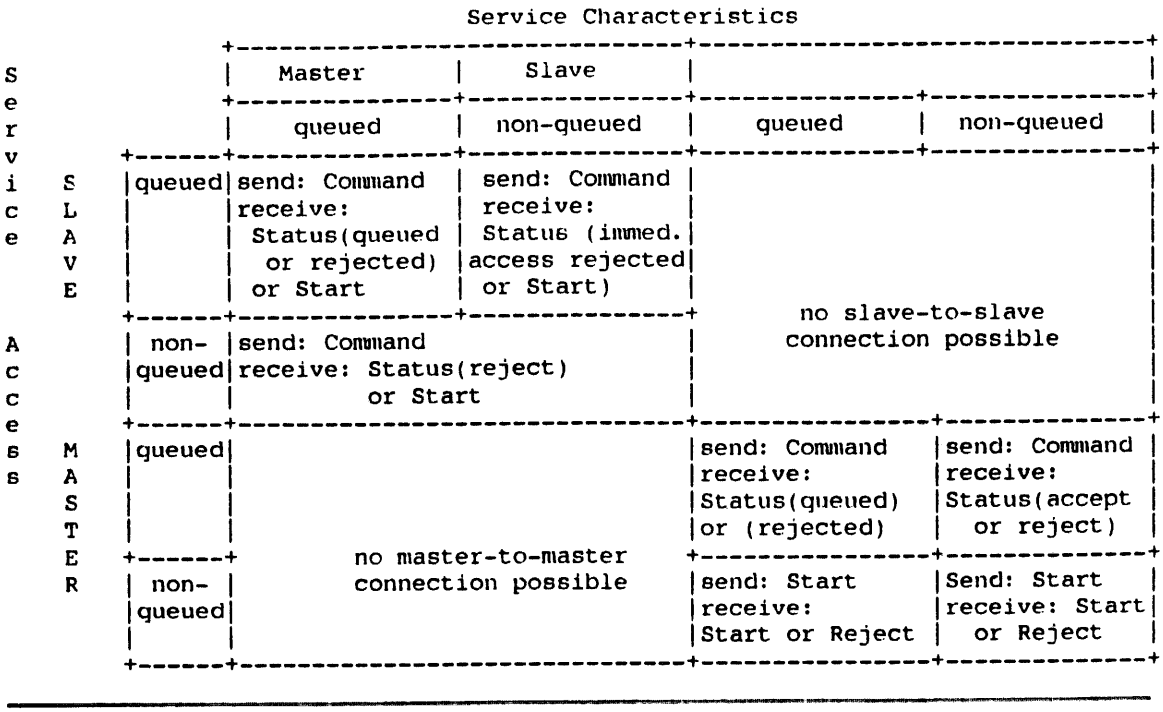


When resources to satisfy an entry in the queue become available on the object node, the corresponding connection can be started. When a master node starts a connection using a Start slot, it supplies a request or entry identifier that was uniquely identified on the node to which the Start slot is directed.

5.1.1.2 Queue Access

Queued and non-queued access methods can be used by a subject to initiate a connection to an object. Access from a subject to an object is established using Start Messages/Start slots and Command/Status messages. Figure 5-2 correlates access methods and service characteristics.

Figure 5-2: Access Methods and Service Characteristics



Note that there is the difference between connection initiation from a master to a slave and connection initiation from a slave to a master. A slave can use only a Command message to provide queued and non-queued access, but a master can provide queued and non-queued access using a Start message/Start Slot as well as a Command message.

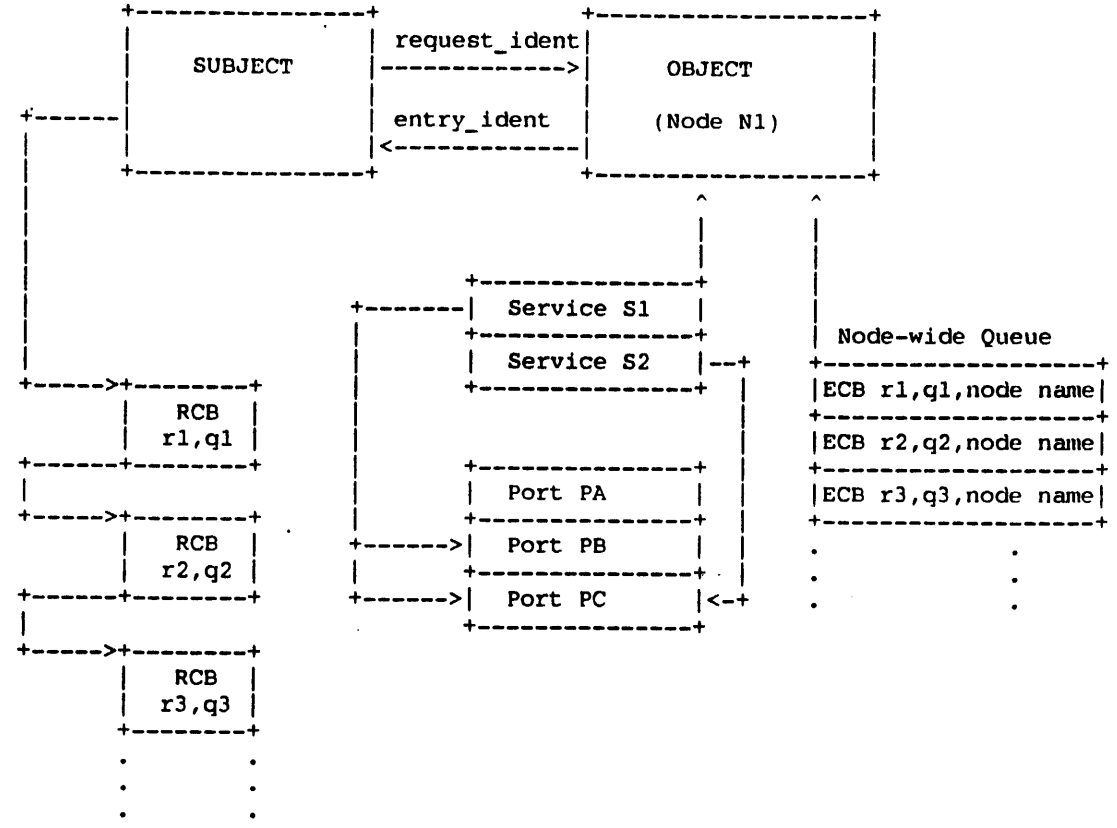
5.1.1.3 Queue Structure

In Figure 5-3 the structure of the request-blocks list on the subject node and the queue-structure on the object node do not attempt to dictate an implementation, but rather serve as an example of request_ident-entry_ident usage. Other parameters that define data used by operations available for queue control are shown united in the Queue Entry Control Block.

The structure of the queue in the object node is assumed to be a node-wide queue to provide uniqueness of the request and entry identifiers among all entries. Actual queuing is done to the services and ports. Queue position parameters attached to each request denote the position of the entry within the queue directed to a particular service and port. Each entry can be designated for a particular port or all ports offering this service. Queuing of requests depends upon the service and port name translation.

In Figure 5-3, the subject stores in the list of the request control blocks (RCB) all requests that are currently being queued. A unique request_ident (rn) is provided for each block in the list. The object has a node-wide queue that includes entries queued to all services. Each entry is represented by a Queue Entry Control Block (ECB). Uniqueness of the entry in the queue is provided by the entry and request identifiers and name of the soliciting node. Command/Status message exchange allows creation of a corresponding ECB entry (qn) for each request (rn). Each ECB in the queue and each RCB are identified by the same pair of request_ident and entry_ident. All entries in the queue are linked into a list that is queued to a specific service. Each entry changes its queue depth moving up in the queue according to the processing of the requests. On the figure below service S1 is offered by the ports PB and PC and service S2 is offered by the port PC.

Figure 5-3: Queue and Request List Structure



5.1.1.4 Queue Operations

LAT provides a set of operations that a subject can perform in order to operate with a service's queue: insert an entry into a service's queue, delete an entry, acquire information about any particular entry in the service queue, etc. The Command message and the Status message contain fields that allow an exchange

of operation codes and parameters between the subject and the object (see LAT Message formats). A subject defines an operation code in the COMMAND_TYPE in the Command message.

Operations performed by a subject can be viewed as "access" operations and "status request" operations which are specified by the COMMAND_TYPE and COMMAND_MODIFIER fields of the Command message. "Access" type operations always define an entry and allow inclusion or cancellation of an individual entry in the queue. "Status request" type operations allow request of the status of the individual or multiple entries in the queue. A Command message type is defined by an operation code (value) and a specifier (bit mask). The following operation codes are specified:

- non-queued access operation
- queued access operation
- cancel entry operation
- individual status operation
- multiple status operation
- queue status operation

There is also a possibility of a "local" operation - an object node can delete an entry in the local queue without request from the subject. In that case, a Status message is sent by the object to the corresponding subject to inform the subject about deletion of an entry.

A side from the local operation described above, any operation on an existing entry in the queue can be performed only by the node that queued the entry. The object node that contains the queue must always validate the subject node-name to verify whether the specified entry was queued by the node issuing the request. If not, the object node must return a Status message with the error reason "Inconsistent or illegal request."

5.1.1.5 Connection Solicitation Operations

"Access" operations allow a subject to request access to a service or delete an entry in the service's queue. Defined access operations are:

- Solicit queued access to the service - the subject supplies a REQUEST_IDENTIFIER and, if the object accepts the request for queuing, it builds and queues an ECB and sends a Status message to the subject. The Status message includes an ENTRY_IDENTIFIER along with other parameters (see "Status Message").

If the queue request is not accepted, the Status message returns with a reject reason.

- Solicit non-queued access to the service - if the request can't be accepted immediately, a Status message with a reject reason is returned by the object.

- Delete an entry in the queue - a subject can cancel an entry in the queue. A subject identifies the entry in the queue using ENTRY_IDENTIFIER and REQUEST_IDENTIFIER values. An object node must verify a subject node name against the name of the node that queued an entry.

When the Command message is sent to the solicited node, the solicited node's decision as to how to queue the request to the service and port is based on the service/port name information supplied in the Command message:

- if only the service name is provided - the request is queued to the service and the first available port offering this service is chosen. An error is returned if the node does not offer the requested service.

- if only the port name is provided - the request is queued to the default service. An error is returned if the requested port does not offer the specified service. The default service name is returned in the SRVC_NAME field of the Status message. A null name (the name length counter = 0) is returned if no default service is defined.

- if both the service and port names are provided - the request is queued to the requested service for the specified port. An error is returned if the node does not support the requested service or the port does not offer the requested service.

- if neither the service nor the port name is provided - the request is queued to the default service on the node and the first available port offering the default service is chosen. The default service name is returned in the SRVC_NAME field of the Status message.

An example of request queuing is presented below (refer to "Name Translation Process") The source node that made a solicitation request translated the source <NODE_NAME><SERVICE_NAME> and sent a solicitation request to the node N1. Table 5-1 shows how the name translation process results in a queue structure on the object node N1 depending upon the given <SERVICE_NAME><PORT_NAME> combinations.

Table 5-1: Name Translation Examples

presented name	entry ident	service queued to	port used	service-wide queue position	node-wide queue position
<S1><PB>	r1,q1	S1	PB	queue_posit = 1	queue_posit = 1
<S1>	r2,q2	S1	undef.	queue_posit = 2	queue_posit = 2
<PB>	r3,q3	S1	PB	queue_posit = 3	queue_posit = 3
none	r4,q4	S1	undef.	queue_posit = 4	queue_posit = 4
<PC>	r5,q5	S1	PC	queue_posit = 5	queue_posit = 5
<S2>	r6,q6	S2	PC	queue_posit = 1	queue_posit = 6
<S2><PC>	r7,q7	S2	PC	queue_posit = 2	queue_posit = 7

Service-wide and node-wide queue_position values give the user the position of an entry by defining its positions in the service and node queues. This is an approximate position and not necessarily an order in which an entry will be taken for processing. An entry is taken for processing when it becomes the highest entry in the node-wide queue for which resources become available.

5.1.1.6 Status Solicitation Operations

An object node returns the status of only those entries that were queued by the subject node issuing a Command message (i.e., an object node has to check an entry against the soliciting node name). There are three types of status solicitation requests as defined by the COMMAND_TYPE operation code:

- Individual entry status - this type of operation requires an object node to return the status of a particular entry to the subject node. Request and entry identifiers both are not 0. Note that the object node can still respond with a Status message containing a multiple-entries status (to provide concatenating of the entries in one Status message).

- Multiple entries status - a subject node can inquire about all queued entries. Request and entry identifiers both must be 0. When this operation is performed, all entries queued by the subject node (and only those) are included in the Status message and sent back by the object node.

- Queue status - a subject node can query the status of the queue. Request and entry identifiers both must be 0. When a message has to be returned by an object node, the ENTRIES_COUNTER field in the Status message must be 0, and no entry information will be included in the message. In the future, extensible parameter fields in the Status message can be used to return different types of information about the queue status on the object node.

As stated above, the solicited node can always concatenate the status of several entries in one Status message in order to minimize message traffic when the soliciting nodes require status reports on a timer or queue-depth change basis (see the following sections). That is, the soliciting node can always receive a Status message that includes more entries than the soliciting node requested. The soliciting node should provide the necessary filtering of entries.

The COMMAND_MODIFIER bit mask specifies how the status of an individual entry is to be sent by an object node. The defined modifiers are the periodic status request and the queue-depth-change status request. The specifier creates a request state for an individual entry. That request state cannot be changed during an entry's life-time in the queue.

An object node that is queuing connection requests specifies the optional parameters, destination port name and destination service name. Those parameters are preserved in the Entry Control Blocks for each entry.

When a subject node requests the status of one particular entry, no filtering is performed because each entry is uniquely identified by request-entry identifiers. When a subject node requests multiple-entries status, an object node can use the destination port and service names to filter entries included in the queue as follows:

- if no destination port or service name is specified, the object node includes in the Status message all entries queued by the subject node.
- if only the destination service name is specified, the object node includes in the Status message only those entries that are queued to the specified service.
- if only the destination port name is specified, the object node includes in the Status message only entries queued to the specified port (the name of the destination port was explicitly specified in a Command message).
- if both the destination service and port names are specified, the object node includes in the Status message only entries queued to the specified port and the specified service.

5.1.1.7 Concatenating The Status Entries

The Status message allows sending of the status of more than one queue entry in one status message. Concatenating multiple status entries in one status message allows minimization of message traffic. The object node can concatenate in one message all entries that were queued by the subject node when:

- The subject node requests multiple-entries status.
- The queue depth changes and several entries have a request-state requiring a status message to be sent to the same subject node.
- The timer expired on an entry with a "periodic" request state, and the status of several entries have to be reported to the same subject node.

One note is necessary about the expected behavior of a subject node when it receives multiple status entries in the Status message. If multiple status entries do not fit in the Status message, the object node can send several Status messages to the subject node. Actually, the subject node can't be sure that all entries are included in the Status message. The rule is that the subject node cannot use the absence of an entry in the Status message to make any conclusions about whether an entry is still pending in the object node's queue. The subject node can time-out the queued state of the entry if no entry status information has been returned in the Status message(s) during the time-out period.

5.1.1.8 Retransmission And Time-out Policies

Since the Command-Status/Start message exchange is going outside virtual circuit context policies that govern event time-outs, timer values and retransmission counters play an important role. Values recommended or required for those timers and counters are presented in the section entitled "Defined parameters and recommended or required default values".

- MULT_STAT_TIMER defines the time interval between Status messages when multiple Status messages have to be sent by the object node to provide information on all queued entries.
- STAT_REP_TIMER is used by an object node as a retransmit interval to report the status of entries in the queue when it is requested by a subject node.
- A node queues an entry using a Command message. The node that receives the request for queuing responds with a Status message (see "State Tables" below). RETR_COMM_TIMER parameter defines time interval for a node to retransmit an unanswered Command message. RETR_COMM_COUNT defines number of times this process must be repeated before timing-out the request.

- When a master node queues an entry to a slave node, the master node has to respond with the Start message upon receiving from the slave node the Status message informing the master that the entry has been chosen for processing (see "State Tables" below). RETR_STAT_TIMER and RETR_STAT_COUNT parameters define a time interval forretransmission of an unanswered Status message from the slave where an entry is queued and number of times that process must be repeated before entry is timed-out.

5.1.2 Connection Initiation

On the virtual circuit level a connection can be established only from a master to a slave. That is, a master always behaves as a subject and a slave always behaves as an object. Implemented connection solicitation process allows connection initiation from slave nodes to master nodes and provides communication of data to the master nodes (and to the application terminals) using the underlying LAT protocol. The connection initiation process allows arbitration of conflicting requests for use of services by queuing concurrent requests, acquiring information about queued requests, and canceling a solicitation. Because of the asymmetrical nature of the LAT architecture, the connection solicitation process allows slaves to initiate connections to services offered by masters by requesting a master to actually start a connection.

To:

- preserve the investment in the slave and master node implementations
- allow shared services offered by slave and master nodes to be arbitrated
- allow slave node application processes to initiate sessions to master node ports
- multiplex all data over a single virtual circuit
- provide access to the status of queued processes

The LAT architecture allows both slaves and masters to behave as subjects and objects (i.e., both slave and master nodes can "solicit" connections to services).

The LAT architecture allows slaves to use the connection solicitation process to connect to masters. Correspondingly, the masters that support this version of the architecture also can use the connection solicitation process to arbitrate connection requests and acquire queue information about the slaves that support this version of the architecture Depending upon the status of the node that offers the service and the service characteristics, a master can directly connect to a slave or initiate connection by using the connection solicitation process. See Figures 6-4

through 6-7 for connection initiation examples and Tables 6-2 through 6-6 for the state tables.

LAT implements two messages to allow slave-initiated connections to masters - a Command message and a Status message. Both messages are physically addressed. Once the decision is made by the solicitor to establish a connection using the solicitation process, it formats and sends a Command message to the node that provides the service. Such a Command message (specified below) informs the solicited node of the solicitor's desire to use the service. As discussed before, this message has two additional uses: a) to cancel a previous solicitation, and b) to inquire about service status and queue position.

The connection solicitation mechanism allows a slave node to solicit a connection across an already existing virtual circuit made in the "wrong" direction. In order to do that, the slave node should know that such a virtual circuit already exists. SLAVE_NODE_NAME and MASTER_NODE_NAME fields contained in the Start message allow the soliciting node to identify the name of the node connected to the virtual circuit. That makes it possible to solicit a connection using an already established virtual circuit.

5.1.2.1 Solicitation Process Message Flow

To clarify how the node status and virtual circuit direction influence connection solicitation, some examples of establishing a connection between master and slave nodes are shown in Figures 6-3 through 6-5. the service that is being connected to possesses "queued" characteristic. Message flow diagrams are presented together with explanations to clarify those examples. In the message flow diagrams, Start messages are not shown. In these examples, rN means the request_identifier N, which is assigned by a soliciting node, and qN means the entry_identifier N, which is assigned to the request by a solicited node.

Initiation:

How connection is initiated between nodes depends upon service characteristics, the method of initiating connections to the node (inbound/outbound bits in the NODE_STATUS field of the Response message) and the direction of the already established virtual circuit (see Tables 5-2 through 5-5 for the connection solicitation state tables). The rules that define possible access methods to services are described in Figure 5-2.

Queuing:

If the service is already in use, the solicitor may still attempt to initiate a connection to the service. The request may be placed in a queue of requests to the service for servicing at a later time, provided that the service characteristics permit such queuing. Note that a mechanism exists to cancel a previously issued solicitation should the solicitor later decide not to use the service.

Upon receipt of the Command message, the Master replies with a) a Status Message or b) a Start Message or c) a Run Message containing a Start Slot.

The Status message is sent by a master only in two cases: if the solicitation is rejected by the node or if the solicitation is accepted but the service is currently being used by some other process and the service characteristics allow queuing of requests. Upon receipt of the Command message, a slave always responds with the Status message.

The Status message indicates the acceptance of the solicitation if the service is busy with another user, the service possesses characteristics that permit queuing, and the user requested "queued" access. In this case, a "queue-entry" identifier is assigned by the solicited node and the solicitation request is placed in the node queue. This entry identifier and its position in the queue are passed back to the soliciting node in the Status Message.

The solicitor has the ability to inquire about the status of queued entries. The Command message with a certain COMMAND_TYPE (see "Command Message") is issued by the soliciting node to perform this inquiry. The solicited node responds with a Status Message supplying information about queued entry.

Rejection:

The Status message returned by a solicited node may indicate rejection for a variety of reasons; for example, a resource problem at the solicited node, authorization failure, the service is busy and its characteristics do not support queuing, etc. See the Status message rejection codes.

Rejection of the solicitation request by the solicited node causes the soliciting node to delete the corresponding request from its context. It also can happen that the request from the soliciting node has been accepted and the solicited node responded with a Start slot, but the soliciting node may decide not to use the connection. A typical reason may be that the solicitor has insufficient resources to complete the connection. In this case, the soliciting node deletes the request from its context and sends a reject slot that causes the solicited node also to delete the corresponding entry in its queue.

Note that the solicited node itself also can delete a queue entry created by a solicitation request from a soliciting node. The solicited node in that case should send a Status message with a reject reason back to the solicitor and the solicitor will delete the request from its context.

Acceptance:

If the service on a master node can accept a connection request immediately, no Status message is returned on success. Instead, the master attempts to establish the connection between the soliciting and solicited nodes. The underlying virtual circuit may not already exist between the master and the slave nodes. If it already exists, the connection uses the existing virtual circuit and a Run message containing a Start slot is sent by the master to the slave node. If it doesn't exist, the virtual circuit must somehow be established. There is an inherent asymmetry in the virtual circuit establishment process as defined by the LAT Architecture, namely that the establishment must be initiated by the master. Consequently, the master must initiate the creation of the virtual circuit by sending a Start message to the slave node.

The service on a slave node always responds with a Status message to the master's request. The master node can see whether the request is queued from the ENTRY_STATUS bit in the Status message. When the entry reaches the top of the queue, the slave node sends a Status message with the "entry accepted for processing" bit set in the ENTRY_STATUS field of the Status message. When the master node receives a Status message with this bit set, the master node can initiate a connection using a Start slot or create a virtual circuit using a Start message. An attempt by a master node to start a connection on a queued entry will be rejected by a slave node using a Reject slot with a "request is queued" error code.

Resolicitation:

Should a Command message from the subject node become lost in transmission to the object node, the solicitor can retransmit the Solicit Message at a rate defined by the corresponding timer. No problem can arise from duplicate requests being received by the solicited node on behalf of the same session because each request is 'tagged' with the request_ident assigned by the solicitor. Only one Command message from a given soliciting node with a given request_ident is allowed.

Resolicitation may be also caused by events such as the following:

1. the subject node queues the request to the object node;

2. the subject node crashes;

3. the subject node immediately restarts while request is still in the queue on the object node.

If the subject node queues a new entry using the same request identifier that was used for the entry which is still queued on the object node, the object node has no way of knowing that it is actually a different request; the second request may specify a totally different object. If this connection is allowed to be started, unintended results may occur. For example, ASCII text may be sent to a graphic plotter instead of a printer.

To avoid this situation, the service and port names in the Command message, Status message and Start slot must be consistent in order to start a connection.

If the object node receives a Command message specifying the same request identifier as an existing queue entry and the same object is specified (service and port names are the same), the object node returns a Status message with the reason code "request already queued". If the object specified is different, then the object node deletes the existing entry from the queue, queues new entry, and then sends a Status message specifying success.

When the object is available and the connection starts, all parameters of the queued entry (service and port names) are included in the Start slot by the muster node and verified against stored parameters of the request by the slave. If this verification fails, the error "solicitation request is corrupted" is returned to the user and the entry is deleted from the queue as shown in Table 5-2 below.

Table 5-2: Example of Connection Resolicitation

Subject (Master)		Object (Slave)	Action
	command msg (access r1,0) ————————->		Subject queues an entry. Entry is accepted and queued. All parameters saved
	status msg (r1,q1) (accepted) <————————		Status message is not received by the subject or subject went down, came up again and accidentially chooses the same identifier
	command msg (access r1,0) ————————->		Subject queues an entry. Object verifies parameters.

Table 5-2 (Cont.): Example of Connection Resolicitation

Subject (Master)	Object (Slave)	Action
	status (r1,q1) (already queued) <————————·	Parameters match: Object returns status "entry already queued"
		Parameters don't match: Object deletes old entry and queues new one. All parameters saved.
	status msg (r1,q1) (success) <————————·	Status message returned.
	status msg (r1,q1 ready) ————————·>	When resourse is available slave sends status message
	start slot (r1) ————————·>	Subject (master) starts connection.
	start (accept/reject) <————————·	Object verifies parameters in the Start slot against stored parameters of the entry and responds with start (if verification succeeds) or reject.
	command msg (access r1,0) ————————·>	Subject queues an entry. Entry is accepted and queued. All parameters saved
	status msg (r1,q1) (accepted) <————————·	Status message is not received by the subject or subject went down, came up again and accidentially chooses the same identifier
	command msg (access r1,0) ————————·>	Subject queues an entry. Object deletes old entry and queues new one. All parameters saved.
	status msg (r1,q1)(entry already queued) <————————·	Status message returned.
	start slot (r1) <————————·	Object (master) starts connection. Subject verifies parameters in the Start slot

Table 5–2 (Cont.): Example of Connection Resolicitation

Subject (Master)		Object (Slave)	Action
	start (accept/reject) ————————·>		against stored parameters of the entry and responds with start (if verification succeeds) or reject.

Table 5–3: Example of Slave Initiating Connection to Master

Slave	Master	Action
	command msg (r1,0) ————————·>	Slave sends command message. Service is available. Master responds with Start message establishing connection between master and slave.
	start slot (r1) <————————·	No queue has been established for that request.
	start/reject slot ————————·>	Slave responds with Start or Reject.
	command msg (r1,0) ————————·>	Slave sends command message. Solicitation is rejected.
	status msg (r1,0) (reject reason) <————————·	No queue entry has been established for that request.
	Command msg (r1,0) ————————·>	Slave sends command message. Solicitation is accepted. Service is busy and queued. Queue entry created,
	Status msg (r1,q1,) ————————·>	entry_ident and queue position returned to soliciting node.
	.	
	.	
	.	
	Command (r1,q1) (status) ————————·>	Slave can solicit again to inquire queue status.
	Status (r1,q1) (status) <————————·	Master responds with status information.
	.	
	.	
	.	Several solicitation requests can be queued uniquely identified by r2q2,r3q3, etc.
	.	

Table 5–3 (Cont.): Example of Slave Initiating Connection to Master

Slave	Master	Action
	.	
	.	
	Start slot (r1) <———————-	When reaches top of the queue, master sends Start.
	Start/reject slot ———————->	Slave responds with Start or Reject.
	Command msg (r1,0) ———————->	Slave sends command message. Solicitation is accepted. Service is busy and queued. Queue entry created.
	Status msg (r1,q1) ———————-	Entry_ident and queue position returned to soliciting node.
	.	
	.	
	Status msg (r1,0) (reject reason) ———————->	Some time later solicited node decided to delete entry from the queue and reports that to solicitor.

Table 5–4: Example of Master Initiating Connection to Slave

Master	Slave	Action
	Start slot ———————->	Master establishes direct connection to the slave by sending a Start slot.
	start/reject slot <———————-	Slave accepts request (if queue is empty) or rejects it (if queue is busy).
	Command msg (r1,0) <———————-	Master solicits queued connection to the slave.
	Status msg (r1,q1) <———————-	Slave returns Status msg. Master can send Command message again or request slave to send entry status periodically or when queue depth change.

Table 5–4 (Cont.): Example of Master Initiating Connection to Slave

Master	Slave	Action
	.	
	.	
	Status msg (r1,q1) ——————->	
	.	
	.	
	.	
	Status msg (r1,q1) <——————-	When entry reaches top of the queue, Status message reports this event to the Master.
	Start slot (q1) ——————->	Now master can start connecttion by sending Start slot.
	Start/reject slot <——————-	Slave responds with Start or Reject slot.

Table 5–5: Example of Connection Initiation Between Nodes Operating in Master/Slave Mode

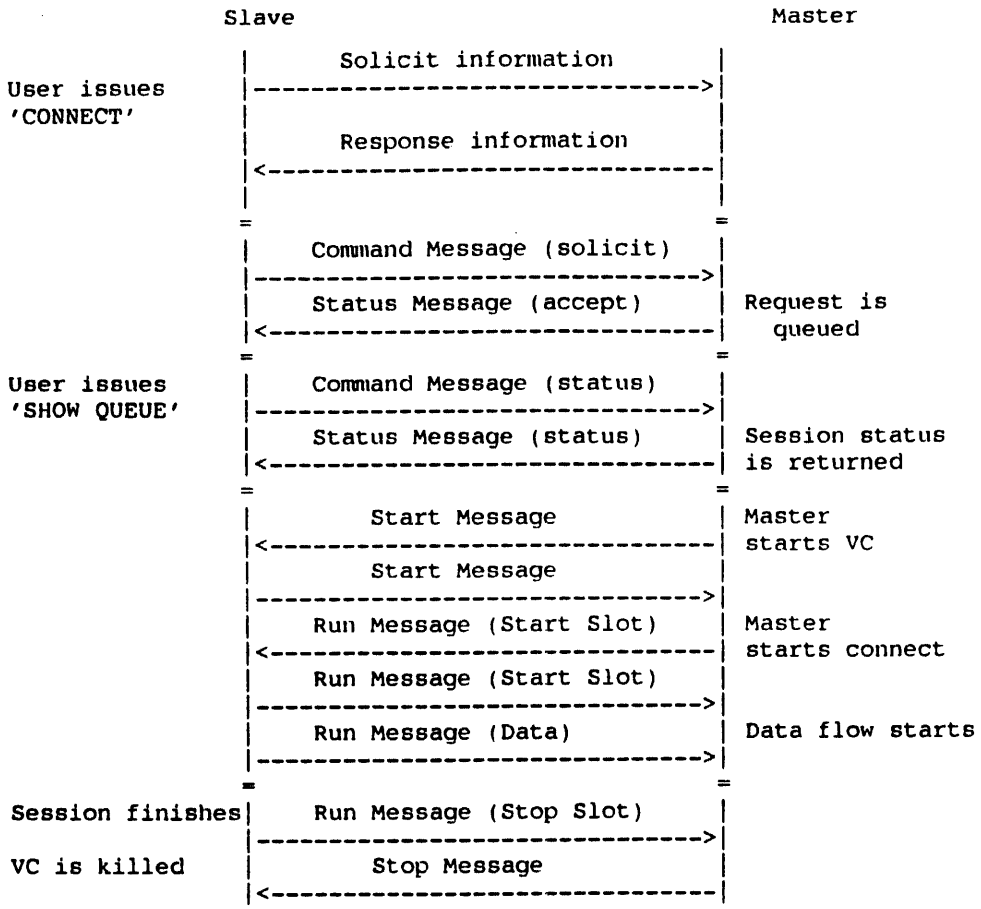
Master/Slave	Master/Slave	Action
	virtual circuit direction ——————->	Virtual circuit already established as shown. Connection can be established in the same direction using solicitation or/and direct connection.
	Command msg (r1,0) ——————->	Master can solicit queued connection from the slave.
	Status msg (r1,q1) <——————-	Slave can return Status (accept) message.
	.	
	.	
	Status msg (r1,q1) (chosen) ——————->	When Status message with "entry is chosen" comes back

Table 5–5 (Cont.): Example of Connection Initiation Between Nodes Operating in Master/Slave Mode

Master/Slave	Master/Slave	Action
	Start slot (q1) ————————->	Master can start a a connection.
	Start/reject slot ————————->	
	virtual circuit direction ————————->	Virtual circuit already established as shown. Connection may be established in reverse direction on the same VC using solicitation.
	Command msg (r1,0) <———————-	Solicitation is accepted. Service is busy and queued.
	Status msg (r1,q1) <———————-	Queue entry has been created entry_ident and queue position returned to soliciting node.
	.	
	.	
	Start slot (r1) <———————-	When request can be satisfied, start sent.
	Start/reject slot ————————->	Start/reject sent.

Table 5-5 presents one specific example of the message flow for a slave node soliciting a print service on a master node that has other sessions already queued to the service.

Figure 5-4: Exchange Between Slave and Master



The above diagram shows an exchange between the slave and the master in which the slave solicits the service, the master accepts the solicitation, and the slave later accepts the connection to the service.

5.1.2.2 Solicitation process state-tables

The solicitation process exists simultaneously as a position on a state-table on both ends of the connection - the soliciting node and the solicited node. Events on one end change the state on both ends of the connection. Soliciting and solicited node state-tables are shown in Tables 6-6 through 6-10.

Notes:

- The timeout algorithm presumes a retransmit_timer and a retransmit_count or timeout and global_timeout events to avoid looping within the soliciting state. Timers should start every time a state changes. If a Status message

or a Start message is not received when the retransmit_timer expired, a retransmit_counter (or a global_timer) can be used to repeat that process. If that fails, the solicited node is presumed to be down.

- The state tables below describe the establishing of a session through the solicitation and queuing process. There is a transition from these tables to the Host/Server slot state tables presented in the previous chapter entitled "Circuit and session layers". "Next state" in this case references to corresponding state in one of the host or server slot state tables in the sections named "Host Slot State Table and "Terminal Server Slot State Table".

- Note that the entry taken for processing must stay in the queue. This entry is deleted from the queue and the request-entry identifiers reused only after the session is started.

- To simplify the state transitions, multiple status messages are not shown. Transition from one state to another is made based on the presence (absence) of a particular entry status.

- When a resource allocation failure happens, the node should make its best attempt to inform a partner about the allocation failure.

- Request and entry identifiers can not be recycled faster then the node consumes the queue requests. That means the node is limited to 65534 outstanding requests.

- In the state tables below, a value of 0 is used when no identifier exists.

- The request and entry identifiers can not be zero.

Table 5–6: Subject (Slave) Node State Table

State	Event	Action	Next State
halted	user solicit	send Command (access r1,0) create an entry (r1)	soliciting
	Status received (r1,q1)	no action	halted
	Start received (r1)	send Reject	halted
	any other user event	return error to user	halted
soliciting	timeout	send Command (access r1,0)	soliciting
	global timeout	delete entry (r1)	halted

Table 5–6 (Cont.): Subject (Slave) Node State Table

State	Event	Action	Next State
Soliciting (cont.)	Status received (accept entry r1,q1)	update queue state	queued
	Status received (reject entry r1)	delete entry (r1)	halted
	Start received (r1)	Verify parameters	
		if match: delete queue state session starts	start-rcv Table 5-4
		if not match: delete queue state	send reject (corrupted entry)
	User cancel entry (r1)	update queue state	cancel
queued	user cancel entry (r1)	send Command (cancel r1,q1)	cancel
	user status request (r1)	send Command (status r1,q1)	queued
	entry timed out (r1) (no status returned)	delete entry (r1)	halted
	Status received (queued r1,q1)	update queue state	queued
	Status received (r1,q1 deleted)	delete entry (r1)	halted
	Start received (r1)	Verify parameters	
		if match: delete queue state (session starts)	start-rcv Table 5-4
		if not match: delete queue state	send reject (corrupted entry)
cancel	Status received (r1,q1 deleted)	delete entry (r1)	halted
	timeout	send Command (cancel r1,q1)	cancel
	global time out	delete entry (r1)	halted
	Status received (r1,q1 accepted)	send Command (cancel r1,q1)	cancel
	Status received (r1 rejected)	delete entry (r1)	halted

Table 5–6 (Cont.): Subject (Slave) Node State Table

State	Event	Action	Next State
Cancel (cont.)	Start received (r1)	send Reject delete entry (r1)	halted

Table 5-7 describes the case where the host (slave) node attempts to queue a request to the queued service offered by the server (master) node.

Table 5–7: Object (Master) Node State Table

State	Event	Action	Next State
halted	Command received (access r1,0)	send Status (queued r1,q1 create an entry) (r1,q1)	queued
		send Status (reject r1)	halted
		send Start (r1)	connect_req Table 5-3
	other Command message received (r1q1)	return Status (r1,0) (unknown entry)	halted
	any other user event	return error to user	halted
queued	Command received (cancel r1,q1)	send Status (r1q1 deleted) delete entry	halted
	Command received (status r1,q1)	send Status (r1,q1)	queued
	Command received (access r1,0)	Verify parameters	
		if match: Send Status (already queued r1,q1)	queued
		if don't match: delete old, queue new Send Status (queued r1,q1)	
	send status event (timer or queue-depth)	send Status (r1,q1) (r1,q1 deleted) delete entry	halted
	user delete entry (r1)	send Status (r1,q1 deleted) delte entry	halted

Table 5–7 (Cont.): Object (Master) Node State Table

State	Event	Action	Next State
Queued (cont.)	resources available	send Start (r1) delete queue state	connect_req Table 4-3
	resources available can't start	send Status (delete r1,q1) delete entry	halted

Table 5-8 describes a case, where the server (master) acts as an object, and receives a request to queue an entry to one of the offered services.

Table 5–8: Subject (Master) Node State Table

State	Event	Action	Next State
halted	user solicit	send Command (access r1,0) create entry (r1)	soliciting
	any other user request	return error	halted
	Status received (r1,q1)	no action	halted
soliciting	timeout	send Command (access r1,0)	soliciting
	global timeout	delete entry (r1)	halted
	Status received (accept entry r1,q1)	update queue state	queued
	Status received (reject entry r1)	delete entry (r1)	halted
	user cancel entry (r1)	update queue state	cancel
queued	user cancel entry (r1)	send Command (cancel r1,q1)	cancel
	user status request (r1)	send Command	queued
	entry timed out (no status returned)	delete entry (r1,q1)	halted
	Status received (queued r1,q1)	update queue state	queued
	Status received (r1,q1 deleted(	delete entry (r1,q1)	halted

Table 5–8 (Cont.): Subject (Master) Node State Table

State	Event	Action	Next State
Queued (cont.)	Status received (process entry r1,q1)	send Start (r1) (session starts) delete queue state	connect_req Table 5-3
cancel	Status received (r1,q1 deleted)	delete entry (r1,q1)	halted
	timeout	send Command (delete r1,q1)	cancel
	global time out	delete entry (r1,q1)	halted
	Status received (r1,q1 accepted/queued)	send Command (cancel r1,q1)	cancel
	Status received (r1 rejected)	delete entry (r1,q1)	halted

Table 5-9 describes the case where server (master) node acts as a subject and attempts to queue a request for queued services offered by the host node (slave).

Table 5–9: Object (Slave) Node State Table

State	Event	Action	Next State
halted	Command received (access r1,0)	send Status (queued r1,q1) create an entry (r1,q1)	queued
		send Status (rejected r1)	halted
	other Command message received (r1,q1)	send Status (r1,0) (unknown entry)	halted
	any other user event	return error	halted
	Start received (r1)	send Reject	halted
queued	Command received (cancel r1,q1)	send Status (r1,q1 deleted) delete entry	halted
	Command received (status r1,q1)	Verify pareameters	

Table 5-9 (Cont.): Object (Slave) Node State Table

State	Event	Action	Next State
Queued (cont.)		if match: Send Status (already queued r1,q1)	queued
		if don't match: delete old, queue new Send Status (queued r1,q1)	
	send status event (timer orqueue-depth)	send Status (r1,q1) update queue state	queued
	user delete entry (r1)	send Status (deleted r1,q1) delete entry	halted
	resources available	send Status (process entry r1,q1)	ready
ready	timeout	send Status (process entry r1,q1)	ready
	global timeout	send Status (deleted r1,q1) delete entry	halted
	Start received (r1)	Verify parameters	
		if match: delete queue state (session starts)	start-rcv Table 5-4
		if not match: delete queue state	send reject (corrupted entry)

Table 5-10 describes a case where a host node (slave) offers services that posess queued characteristics. Node acts as an object and receives a request to queue an entry to one of the services it offers.

5.1.2.3 Name And Information Field Presentation

A subject that requests usage of the resources on an object initiates an exchange of Command-Status messages, and Start(initiate)-Start(response) slots. Those messages contain object names (nodes, services, ports) that are translated, and subject description fields designated for informational purposes. Table 5-10 shows

the rules that subject and object nodes should follow in filling those fields in order to provide name translation and information caching support.

Table 5-10: Name and Information Fields

Messages Flow	Name and Information Fields
Start (init) ⟶	object name fields (source) subject information fields
⟵Start (resp)	object name fields (translated) subject information fields - 0 on send, ignored on receive.
Command ⟶	unique request identifier object name fields (source) subject information fields
⟵Start (init)	request identifier object name fields (translated) subject information fields - 0 on send, ignored on receive.
Start (resp) ⟶	name/information fields must be 0 on send, ignored on receive.
Command ⟶	unique request identifier object name fields (source)
⟵Status	unique entry identifier object name fields (translated) optional subject information fields
Start (init) ⟶	entry identifier object name fields same as in the received Status message
⟵Start (resp)	name/information fields must be 0 on send, ignored on receive.
Command ⟶	unique request identifier object name fields (source) subject information fields
⟵Status	unique entry identifier object name fields (translated) optional subject information fields
⟵Start (init)	request identifier object name fields same as in the sent Status message
Start (resp) ⟶	name/information fields must be 0 on send, ignored on receive.

5.2 Message Formats

This section presents formats, layouts and contents of the messages used by the connection solicitation mechanism.

Node may receive an unsolicited LAT message (Command) of maximum length and is responsible for allocating enough buffers to accomodate it. Command message contains a DATA_LINK_RCV_FRAME_SIZE field that defines the maximum size of the solicited LAT message (Status) that can be send to this node.

5.2.1 Command Message

A Command message is a physically addressed message used to initiate a connection from the slave to the master node and to provide queued services. Figure 5-5 presents the format of the Command message. A detailed description of each field in the message follows.

Figure 5–5: Command Message Format

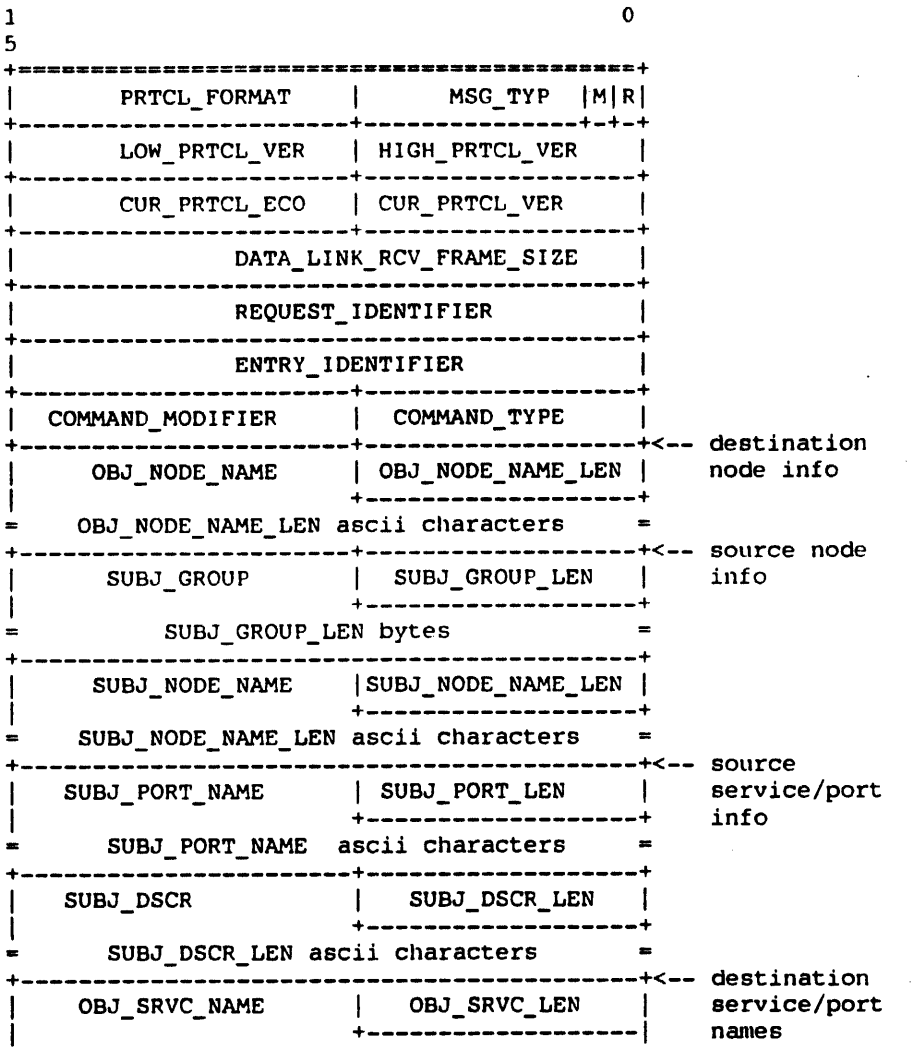
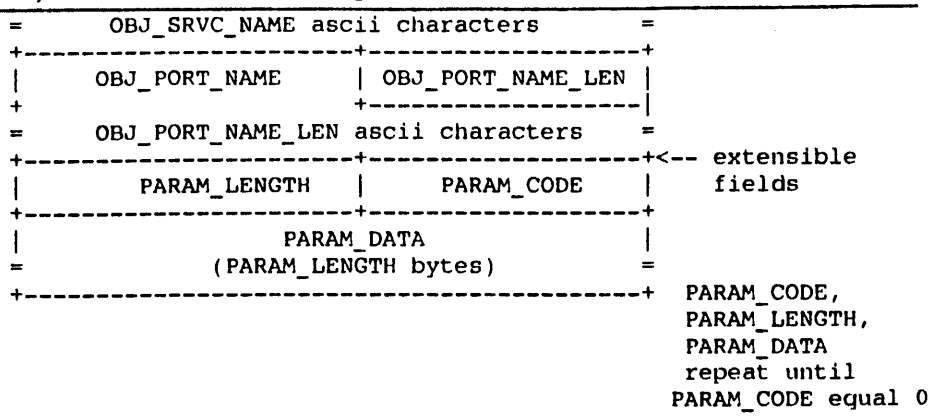


Figure 5–5 Cont'd. on next page

Figure 5-5(Cont.): Command Message Format



- R and M (2 bits) - Must be 0.
- MSG_TYP (6 bits) - Fixed at 12. (Command message).
- PRTCL_FORMAT (1 byte) - Protocol Format flag.
  - Bits 0 through 7 - Must be 0 on transmit; ignored on receive.
- HIGH_PRTCL_VER (1 byte) - Highest protocol version supported by the node.
- LOW_PRTCL_VER (1 byte) - Lowest protocol version supported by the node.
- CUR_PRTCL_VER (1 byte) - Protocol version of this message (current version is 5).
- CUR_PRTCL_ECO (1 byte) - ECO level of CUR_PRTCL_VER for this message (current ECO is 1).
- DATA_LINK_RCV_FRAME_SIZE (2 bytes unsigned) - Maximum size of the LAT message that can be sent to this node. Actual length of a LAT message is DATA_LINK_RCV_FRAME_SIZE-18.
- REQUEST_IDENTIFIER (2 bytes unsigned) - Request identifier. This field contains a solicit request identifier that is assigned by the node soliciting the connection to the service. This value is used by the soliciting node to correlate Status messages arriving from the solicited node with Command messages sent by the soliciting node.
- ENTRY_IDENTIFIER (2 bytes unsigned) - Entry identifier. This field contains the identifier of a previously issued Command message (i.e., one in which COMMAND_TYPE was set to 2).

- COMMAND_TYPE (1 byte unsigned) - Command message operation code. The Command message is issued by the solicited node for various reasons which are summarized by the codes below as a set of values:

  - value = 1 (access) - Solicit non-queued access to the service. If the service cannot immediately satisfy that request, an error is returned; otherwise connection initiation is attempted.

  - value = 2 (access) - Solicit queued access to the service. This value is passed if the message is used to solicit queued access to the service. The request may be queued if the service is busy. The service must have "queued" characteristics, otherwise an error is returned. This command requires REQUEST_IDENTIFIER to be non-zero.

  - value = 3 (access) - Cancel entry in the queue. This value is passed if the message is used to cancel an entry in the queue that was created as a result of a previously issued solicitation (Command message with COMMAND_TYPE set to 2). This command requires ENTRY_IDENTIFIER and REQUEST_IDENTIFIER values to be non-zero.

  - value = 4 (status) - Send status of the entry. This value requires the solicited node to return a Status message with information about an entry queued as a result of a previously issued solicitation (Command message with COMMAND_TYPE set 2). ENTRY_IDENTIFIER and REQUEST_IDENTIFIER must not be 0.

  - value = 5 (status) - Send status of the queue. The solicited node returns information about its node queue in the Status message. ENTRY_IDENTIFIER and REQUEST_IDENTIFIER must be 0.

  - value = 6 (status) - Send status of multiple entries. An object node will return all entries queued by the soliciting node. ENTRY_IDENTIFIER and REQUEST_IDENTIFIER must be 0.

- COMMAND_MODIFIER (1 byte unsigned) - Bit mask that specifies how the status message should be sent by the solicited node. The meaning of the bits (when set)is:

  - bit 0 - Send status of the entry (entries) periodically. This value requires an object node to periodically send the status of the entry in the Status message. This bit can be used with COMMAND_TYPE operation code equal to 2.

  - bit 1 - Send status of the entry (entries) every time the queue depth changes. This value requires the solicited node to return a Status message every time the node-wide queue depth changes. This bit can be used with COMMAND_TYPE operation code equal to 2.

- bits 2-7 - Must be zero.

The next fields represents subject and object node information.

- OBJ_NODE_NAME_LEN (1 byte unsigned) - Length of the next field. A byte containing the length of the OBJ_NODE_NAME field in bytes. A value of zero is illegal.
- OBJ_NODE_NAME (OBJ_NODE_NAME_LEN bytes) - Destination node name. An array of ASCII characters describing the name of the destination node. These characters are constrained as described in the section of the LAT Architecture Specification entitled "Specification of Names."
- SUBJ_GROUP_LEN (1 byte unsigned) - Subject group code byte length. A byte count of the SUBJ_GROUP field. A value of zero is legal and indicates that the subject can access any services. Maximum value is 32 (-> 256 bits).
- SUBJ_GROUP (SUBJ_GROUP_LEN bytes) - Subject group code mask. This field is specified as a bit-mask of up to 256 bits. A bit set to 1 indicates the subject belongs to that group. The first bit of the mask (bit 0) corresponds to group 0. This group code mask represents an Identifier List (IDL).
- SUBJ_NODE_NAME_LEN (1 byte unsigned) - Length of the next field. A byte containing the length of the SUBJ_NODE_NAME field in bytes. A value of zero is illegal.
- SUBJ_NODE_NAME (SUBJ_NODE_NAME_LEN bytes) - Soliciting node name. An array of ASCII characters describing the name of the node issuing the Command message. These characters are constrained as described in the section of the LAT Architecture Specification entitled "Specification of Names."

The next fields represent subject information.

- SUBJ_PORT_LEN (1 byte unsigned) - A byte containing the length in bytes of the SUBJ_PORT_NAME field. A value of 0 means that no port name is provided.
- SUBJ_PORT_NAME (SUBJ_PORT_LEN bytes) - Soliciting node port name. An array of ASCII characters that forms the name of the source port.
- SUBJ_DSCR_LEN (1 byte unsigned) - Subject description length. A byte containing the length in bytes of the SUBJ_DSCR field. A value of 0 means that no description is provided.

- SUBJ_DSCR (SUBJ_DSCR_LEN bytes) - An array of ASCII characters that forms the textual description of the subject.

The next fields represent destination service/port names.

- OBJ_SRVC_NAME_LEN (1 byte unsigned) - Destination service name length. A byte containing the length in bytes of the OBJ_SRVC_NAME field. A value of 0 means that no name is provided.
- OBJ_SRVC_NAME (OBJ_SRVC_NAME_LEN bytes) - Destination service name. An array of ASCII characters that forms the name of the service. This is a service name as advertised previously by the Service message.
- OBJ_PORT_LEN (1 byte unsigned) - Destination service port name length. A byte containing the length in bytes of the next field. A value of 0 means that no name is provided.
- OBJ_PORT_NAME (OBJ_PORT_LEN bytes) - Destination service port name. The solicited node port name requested by the soliciting node.

The next field marks the beginning of the optional information.

- PARAM_CODE (1 byte) - Parameter code. The following codes are defined:
  - Parameter code 0 - Denotes the end of the parameter list.
  - Parameter code 1 - Required service class. PARAM_DATA field specifies service class to be used. If this PARAM_CODE is not present Service Class 1 is requested.
  - Parameter codes 2-127 - Reserved for DEC.
  - Parameter codes 128-255 - Reserved for users.
- PARAM_LEN (1 byte) - Length of the next field in bytes.
- PARAM_DATA (PARAM_LEN bytes) - Parameter data.

5.2.2 Status Message

A Status message is used to return information about acceptance/rejection of the solicitation request. A Status message is physically addressed. Figure 5-6 presents the format of the Status Message. A detailed description of each field in the message follows.

Figure 5-6: Status Message Format

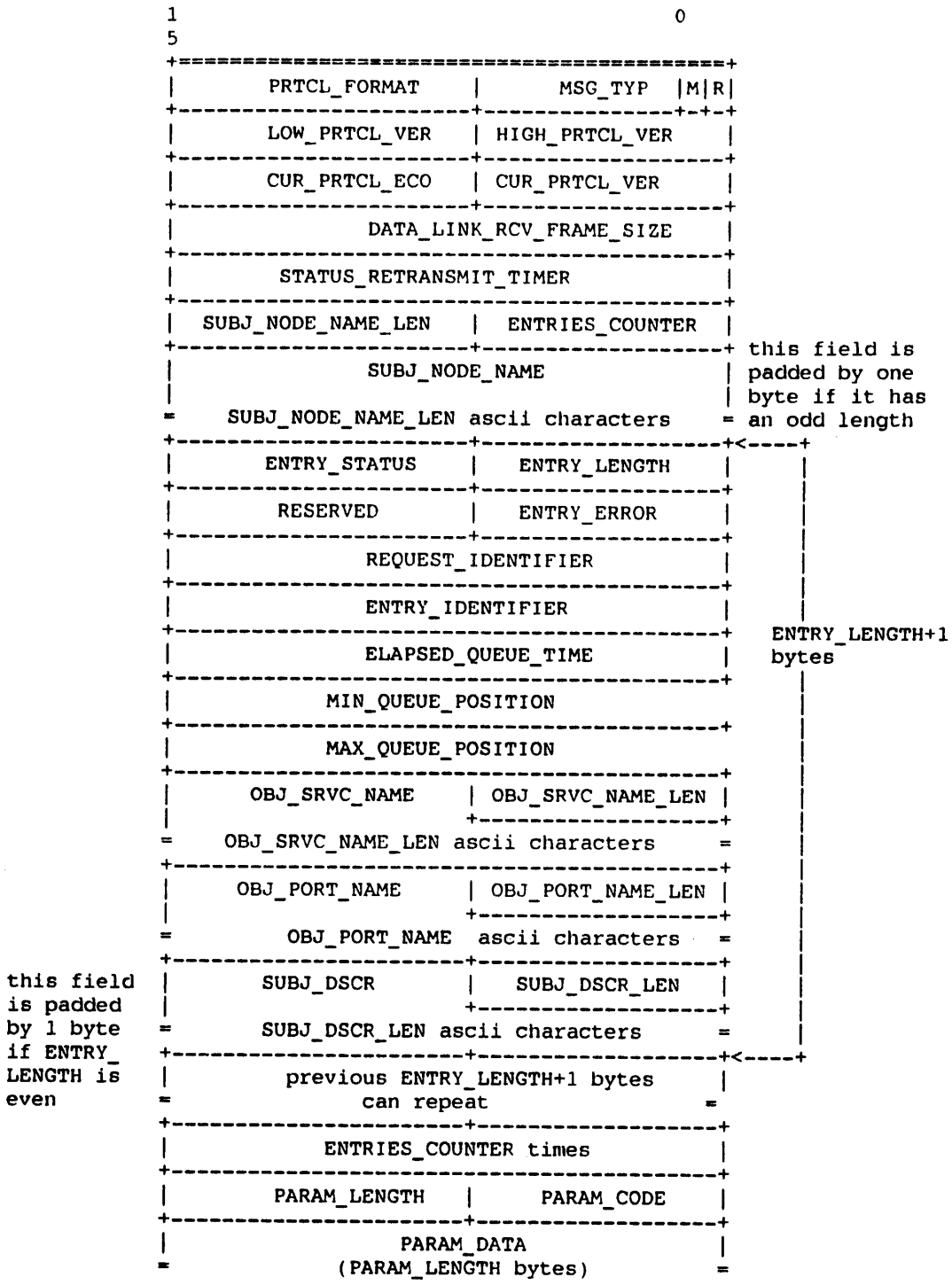
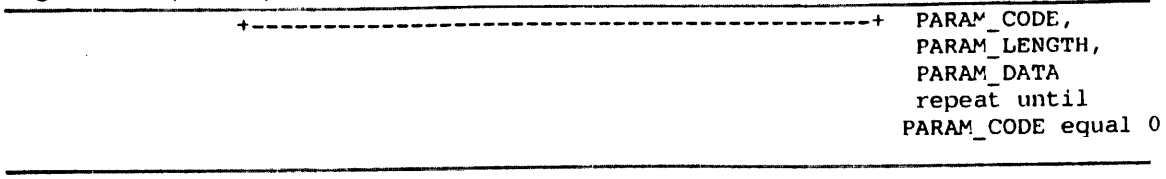


Figure 5-6 Cont'd. on next page

Figure 5-6(Cont.): Status Message Format



- R and M (2 bits) - Must be zero.
- MSG_TYP (6 bits) - Fixed at 13. (Status Message).
- PRTCL_FORMAT (1 byte) - Protocol Format flag.
  - Bit 0 through 7 - Must be 0 on send; ignored on receive.
- HIGH_PRTCL_VER (1 byte) - Highest protocol version supported by the node.
- LOW_PRTCL_VER (1 byte) - Lowest protocol version supported by the node.
- CUR_PRTCL_VER (1 byte) - Protocol version of this message (current version is 5).
- CUR_PRTCL_ECO (1 byte) - ECO level of CUR_PRTCL_VER for this message (current ECO is 1).
- DATA_LINK_RCV_FRAME_SIZE (2 bytes unsigned) - This field must be zero on send, and is ignored on receive.
- STATUS_RETRANSMIT_TIMER (2 bytes unsigned) - Value of status retransmit timer. The soliciting node can request the solicited node to use this value to periodically return the entry status. A value of 0 means that no local timer is supported. The required value is between 10 seconds and 1 hour. The recommended value is 60 seconds.
- ENTRIES_COUNTER (1 byte unsigned) - Number of entries whose status is reported in the message. This field equals 0 for a node queue status request (COMMAND_TYPE equals 5).
- SUBJ_NODE_NAME_LEN (1 byte unsigned) - Length of the next field in bytes. A value of 0 is illegal.
- SUBJ_NODE_NAME (SUBJ_NODE_NAME_LEN bytes) - Name of the node that queued the solicitation request. A string of ASCII characters describing the name of the node that issued the Command message. These characters are constrained as described in the section of the LAT Architecture

Specification entitled "Specification of Names." Note that this field must be padded by one byte if NODE_NAME has an odd length.

The following field marks the beginning of the entry status information.

- ENTRY_LENGTH (1 byte unsigned) - Summary length of all following fields that describe this entry (length in bytes).
- ENTRY_STATUS (1 byte unsigned) - Bit map defining the status of an individual entry. Bit 7 clear means success (the operation is completed with success). Possible additional information is provided by bits 0-6 (when set) as follows:
  - 0 - No additional information is provided.
  - 1 - Request is already queued.
  - 2 - Entry is accepted for processing (sent by slave node when entry is chosen from the queue for processing).
  - 3 - Periodic status return is not supported.
  - 4 - Queue-depth status report is not supported.
  - other values - TBD

  If bit 7 is set, the solicitation request presented in the Command message was rejected, and the following byte contains the rejection reason.
- ENTRY_ERROR (1 byte unsigned) - Solicitation rejection reason. A byte containing a solicitation rejection reason code. This field must be 0 if bit 7 of the ENTRY_STATUS field is clear. If bit 7 of the ENTRY_STATUS field is set, the request is rejected and the ENTRY_ERROR field contains rejection reason values as follows:
  - 0 to 15 - see slot reason codes.
  - 16 - COMMAND_TYPE code is illegal/not supported.
  - 17 - Start slot can't be sent.
  - 18 - Queue entry deleted by local node.
  - 19 - Inconsistent or illegal request parameters.
  - other values - to be defined
- RESERVED (1 byte) - this is reserved byte (zero on send, ignored on receive).

- REQUEST_IDENTIFIER (2 bytes unsigned) - Request identifier. This field contains the identifier of the Command message assigned by the soliciting node. The REQUEST_IDENTIFIER is returned here in order to provide the soliciting node with a means of identifying the solicitation request that is being responded to.

- ENTRY_IDENTIFIER (2 bytes unsigned) - Session identifier. This field contains the identifier of the Command message as assigned by the solicited node. This value must be unique on the node.

- ELAPSED_QUEUE_TIME (2 bytes unsigned) - Elapsed time. The time that this particular entry has been kept in the queue in minutes. When summary information is requested, this value contains ELAPSED_QUEUE_TIME of the active entry ECB. When all bits in this word are set to 1, no time is provided.

- MIN_QUEUE_POSITION (2 bytes unsigned) - Minimum queue position. This value is equal to the entry position in the solicited node service queue. When a summary status of the queue is requested, this value represents number of entries in the service queue.

- MAX_QUEUE_POSITION (2 bytes unsigned) - Maximum queue position. This value is equal to the entry position in the solicited node node-wide queue. When a summary status of the queue is requested, this value represents number of entries in the node-wide queue.

- OBJ_SRVC_NAME_LEN (1 byte unsigned) - Service name length. A byte containing the length in bytes of the next field. A value of 0 means that no name is provided.

- OBJ_SRVC_NAME (OBJ_SRVC_NAME_LEN bytes) - Service name. An array of ASCII characters that forms the name of the service provided by a solicited node.

- OBJ_PORT_NAME_LEN (1 byte unsigned) - A byte containing the length in bytes of the next field. A value of 0 means that no port name is provided.

- OBJ_PORT_NAME (OBJ_PORT_LEN bytes) - Name of the port, provided by a solicited node. An array of ASCII characters.

- SUBJ_DSCR_LEN (1 byte unsigned) - Length of the next field in bytes. A value of zero is legal and indicates that no service description is provided.

- SUBJ_DSCR (SUBJ_DSCR_LEN bytes) - ASCII string of characters representing the textual description of the source service (copied from the SUBJ_DSCR field of the Command message).

The previous field marks the end of the entry status information. One byte should be added after the SUBJ_DSCR field if the ENTRY_LENGTH value is even. If there is more than one entry, the previous fields are repeated ENTRIES_COUNTER times.

- PARAM_CODE (1 byte) - Parameter code. The following codes are defined:
  - Parameter code 0 - Denotes the end of the parameter list.
  - Parameter codes 1-127 - Reserved for DEC.
  - Parameter codes 128-255 - Reserved for users.
- PARAM_LEN (1 byte) - Length of the next field in bytes.
- PARAM_DATA (PARAM_LEN bytes) - Parameter data.

A

Service Class 1 - Interactive And Application Terminals.

While different classes of service share the same underlying data transport service, each class of service defines a different directory service appropriate to the needs of that particular class of service. This service class allows data terminal equipment to be remoted from a host over an intervening Ethernet. Except for the latency associated with reading and writing to the device, the host and terminal server user should find that the remoted terminal performs similarly to a locally connected terminal.

A.1 Local Area Directory Service

The LAT directory service exists to facilitate connections to services within a dynamic Local Area Network by providing an automatic mechanism to map node names, Ethernet addresses, and service names into unique entities. The slot layer of the LAT architecture translates service names into node names and the virtual circuit layer translates node names into 48-bit Ethernet addresses. These translations can be accomplished by utilizing the information provided by the directory service messages. The directory service utilizes the multicast mechanism provided by the Ethernet. It is very responsive to sudden changes in the local area topology.

Implementations are not required to support the directory service messages. A possible alternate strategy which could be implemented by products would be to require that the directory database be entered manually. A significant drawback to this strategy (in addition to the requirement of manual intervention) is that the availability of the service is not known until the connection to it is actually attempted.

The Directory Service for Service Class 1 is supported through the following three messages:

- Service announcement message - a multicast message used by nodes to advertise services.
- Solicit information message - a multicasted or physically addressed message used by nodes to solicit an advertisement.
- Response information message - a physically addressed message used by nodes to respond to the received Solicit information message.

The LAT V5.0 architecture provides the service announcement message to automate the directory service function. Host nodes multicast service announcements at regular intervals while terminal servers listen to these announcements and build a database of services to present to users. The LAT V5.1 architecture supplements the directory service function with the solicit and response information messages. The solicit information message allows nodes which do not listen to multicasted service announcements to still acquire advertisement information without entering the database manually.

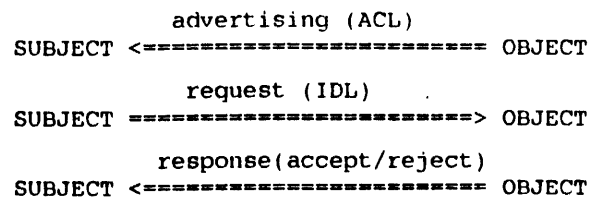
A.2 Service Access Control

Users (subjects, consumers of resources, or active initiators of a connection) may be restricted from accessing certain resources (objects, providers of resources, or passive responders to connections) through the use of group codes. This restriction allows computing resources to be segmented based on such criteria as departmental ownership or physical location. Group codes do not solve security problems.

It is not the intention of the LAT V5.1 architecture to change either the intent or the meaning of group codes as they were defined by the LAT V5.0 architecture. However, the term 'group codes' can be ambiguous and confusing. Therefore, new terminology is introduced with LAT V5.1: Access Control List (ACL) and Identifier List (IDL). Resource (object) group codes are referred to as ACLs and user (subject) group codes are referred to as IDLs.

Figure A-1 illustrates the usage of ACLs and IDLs during the process of connection establish.

Figure A–1: ACL and IDL Flow During Connection Establishment



The algorithm which grants or denies a subject access to an object is based upon comparing ACLs and IDLs. If the intersection between a subject's IDL and an object's ACL is not empty, the subject is granted access to an object. If that intersection is empty, access is denied. Implementations may choose how ACLs and IDLs are managed.

There are three types of group-code fields in LAT messages: node group code(s), service group code(s), and user group code(s). User group codes are IDLs. Service group codes are ACLs. Node group codes is an "OR" function of an IDLs or ACLs group code fields presented in the messages according to the table A-1. The group code field consists of a counter followed by a string of bytes representing a group code mask. This field is specified as a bit-mask of up to 256 bits. A bit set to 1 in a node group code indicates the node belongs to that group (i.e., possesses the corresponding identifier value). The first bit of the mask (bit 0) corresponds to group 0.

LAT V5.1 messages present an IDL or an ACL in a group code field depending upon the message type (i.e., the performed function - advertising or access). Table A-1 defines the group-code fields in the LAT messages.

Table A–1: ACLs and IDLs in Messages

Message	Access-list
Response information	ACL
Multicast Service announcement	ACL
Solicit information	IDL
Command message	IDL
Start slot	IDL

Before a node can solicit the use of a service, the name and characteristics of the service must be made available to the soliciting node, and the soliciting node must have the authority to use the service. As described above, that information can be

acquired through a Response service announcement message which contains the characteristics and status of the service (name, availability, rating, etc.).

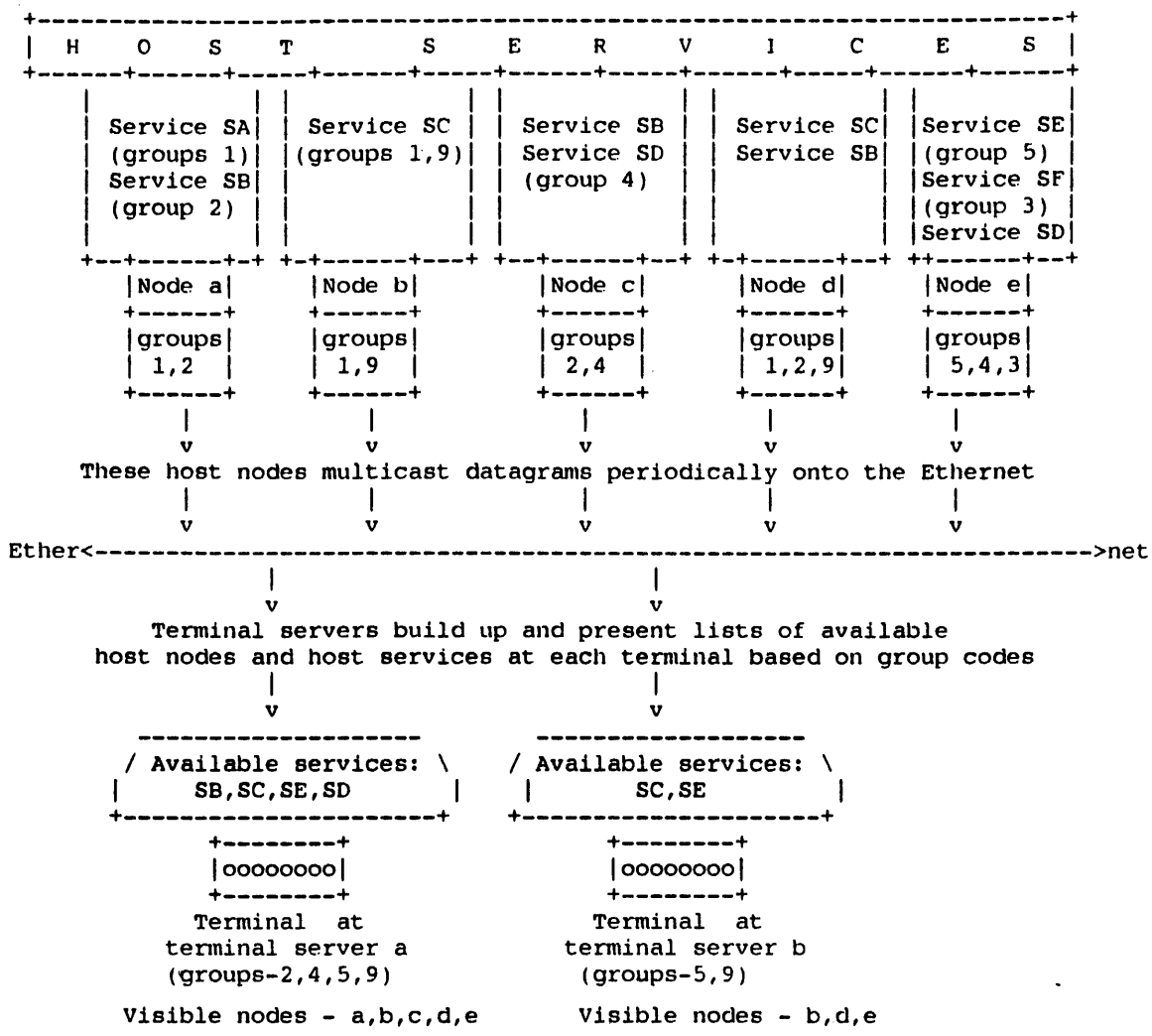
A subject process, no matter how privileged within the context of its own environment, cannot use an object unless the intersection between its IDL and the object's ACL is not empty. In other words, access to a service is allowed if any group code(s) assigned to the solicitor of a service matches any group code assigned to the service on the solicited node.

The set of rules that control service access with the group code mechanism is as follows:

- Nodes ignore Solicit information messages from nodes whose group codes do not intersect their group codes.
- Nodes ignore multicast advertising messages from nodes whose node group code do not intersect their group codes. Therefore a node may provide service announcement message filtering based on group codes.
- If the object node receives a Command message from a subject node whose group code do not intersect with its own group codes, the object node must respond with a Status message specifying "access denied" error code.

The following example illustrates connectivity restriction mechanism of group codes. Services on the host nodes are assigned to one or more groups. Terminals (or terminal servers) are also assigned to one or more groups. Whenever a terminal (subject) and a host node service (object) share the same ACL and IDL (same group code) those two system can interact. For example:

Figure A-2: ACL/IDL Connectivity Restriction Example



The NODE_GROUPS field is a bit mask of 256 bits. If a bit is set, then the node is a member of that group. The first bit of the mask (bit 0) corresponds to group 0. A maximum of 256 groups can be specified (0-255) and therefore this field's maximum length is 32 bytes.

A terminal server implementation should provide a privileged user with a single command which enables all group codes.

A.3 Advertising Services Through Multicast Message

This service class provides a local area directory service to allow users at a terminal server to address host services without the manual intervention of a network manager.

The directory service is very responsive to sudden changes in the local area topology. All terminal servers discover that a particular host node is available within milliseconds of the host node announcing the service. If a host node should crash, the terminal servers can notify the users within a few seconds that the host node may have crashed.

The directory service is based on the multicast mechanism built into the Ethernet.

A.3.1 Host

A.3.1.1 Initialization

A host system manager may specify:

- group codes (R)
- host node name (R)
- host service names and ratings (R)
- the guaranteed minimum Ethernet data link receive buffer size (R)
- the maximum slot size that can be received (R)
- the maximum slot size that can be transmitted (R)
- the physical location of the host node
- a facility number
- host characteristics and status as defined by the particular service class to which the host belongs

It is recommended that the system manager be required to specify none of these parameters in order to communicate with terminal servers that belong to group 0. To accomplish this, LAT host implementations must supply reasonable default values for those parameters labeled (R) (see DEFINED PARAMETERS AND RECOMMENDED OR REQUIRED DEFAULT VALUES).

A host node that has no assigned NODE_NAME or assigned SERVICE_NAME may not announce start of service. Instead, an error should be returned to the system manager.

A.3.1.2 Host Group Codes

Coordination of the access from the certain terminal servers to the host services is provided by usage of the facility-wide group codes arranged by the facility manager.

A.3.1.3 Host Node Names

One or more ASCII names are specified in the multicast message transmitted periodically by each host node.

Although an ASCII names can be up to 127 characters in length, a name should be convenient for a user to remember and type. Hosts should not specify names that are hard (or impossible) for terminal server users to specify. Terminal servers must support a minimum ASCII name length of 16 bytes.

A host node must specify one NODE_NAME in the multicast message.

A host can specify more than eight SERVICE_NAMES, a terminal server is required to buffer a minimum of eight SERVICE_NAMES. A terminal server is required to update all of the information in a multicast message or ignore it.

A.3.1.4 Multiple-Node Service Ratings

In the case where a given service is offered by more than one node, the server selects the node to establish a session based on the service with the highest rating.

A.3.1.5 Steady-State Operation

The host node should periodically multicast the multicast datagram. This interval is measured in seconds and is specified in the HOST_MULTICAST_TIMER field.

Whenever any of the information in the multicast datagram changes, the MSG_INCARNATION must be incremented (modulo 256) and the CHANGE_FLAGS field should reflect which field was changed. The CHANGE_FLAGS field bits are toggled each time the associated field is changed. The flag remains in the new state until the field is changed a second time.

A.3.1.6 System Shutdown

When a host node is "shutting down", the NODE_STATUS field in the multicast datagram should reflect the fact the the host is not accepting new sessions.

If service is terminated by the host system manager, at LEAST one addition multicast message should be transmitted to reflect this change of state.

A.3.2 Terminal Server

A.3.2.1 Initialization

Whenever a terminal server is initialized, and no parameters are supplied interactively, the following default values must be supplied by implementations:

- Group code 0 is enabled.
- ^S and ^Q are used as the output flow control characters
- ^S and ^Q are used as the input flow control characters
- These flow control defaults are used when a slot session is established. Questions related to setting of the flow control characteristics are discussed in the Appendix B of the document.

An implementation might allow a privileged terminal server user to specify:

- group codes (R)
- the guaranteed minimum Ethernet data link receive buffer size (R)
- the circuit timer value (R)
- the maximum slot size that can be received (R)
- the maximum slot size that can be transmitted (R)
- the physical location of the terminal server
- a facility number
- a nickname used to refer to the terminal server
- server characteristic and status as defined by the particular service class to which the terminal server belongs

It is recommended that an implementation not require a privileged terminal server user to specify any of these parameters in order to communicate with host nodes that belong to group 0. In order to accomplish this, an implementation must supply reasonable default values for those parameters labeled (R) (see DEFINED PARAMETERS AND RECOMMENDED OR REQUIRED DEFAULT VALUES).

An implementation of a terminal server might allow all group codes to be enabled with a single command.

A.3.2.2 Building The Circuit Name Database

Each Terminal Server builds a database from the information received in multicast datagrams. A terminal service is require to process all of the information in a multicast datagram, or ignore the datagram.

Terminal servers receive multicast datagrams periodically from each host node. Each time a multicast datagram is received, the terminal servers scan a list of enabled group codes, and if one of the group codes in the multicast datagram matches one of those assigned to the terminal server (user), then the information in the multicast message is added to the local server database (see section on LOCAL AREA DIRECTORY SERVICE). All multicast messages specifying the NODE_GROUP_LENGTH field as zero imply that the host is not offering any services.

If the multicast message contains a NODE_NAME which is not already in the list, a new node entry is created and the information in the multicast datagram is parsed into the entry. If the NODE_NAME has associated SERVICE_NAMES, these too are added to the database.

If the NODE_NAME is already entered, the datagram MSG_INCARNATION field is compared with the MSG_INCARNATION field stored in the list entry to see if the information in this multicast datagram is identical to the data receive previously from the host node. If this field has changed, then the information in the multicast datagram is reparsed into the list entry corresponding to the NODE_NAME. The CHANGE_FLAGS field should be used to reduce the CPU time necessary to parse the entry.

If the NODE_NAME is already entered in the list, but as the message is parsed into an existing entry it is determined that the message was received from a different Ethernet address, the TOTAL_DUPLICATE_NODE_NAME counter should be incremented. The newly received multicast datagram should replace the existing entry. This behavior is possible if a host node has access to more than one Ethernet port and the port that was being used failed. The host node might then start transmitting the same multicast datagram from a different port. Of course, the name could be mistakenly shared by more than one host system. For this

reason, the TOTAL_DUPLICATE_NODE_NAME counter is maintained. The facility manager can diagnose this second aberrant condition by monitoring the node name from a terminal server and observing the changing Ethernet 48-bit address.

Servers maintain the list of all names in memory.

If a terminal server has insufficient memory to buffer all of the received multicast data, NODE_NAME entries are purged from this database in the following order:

1. Timed-out - these are nodes that are known to be unavailable because the LAT_MESSAGE_RETRANSMIT_LIMIT was reached on an active virtual circuit associated with the host node.

2. Unknown - these nodes have stopped transmitting multicast messages for more than 5 times the HOST_MULTICAST_INTERVAL seconds, and are therefore assumed to be in an unusual state (crashed).

3. Shutdown - these are host nodes that have indicated that they are no longer accepting new virtual circuits.

4. Reachable (optional) - these are host nodes that are available, but no virtual circuit is currently established to the node NODE_NAME. If these names are purged from the server database, an unacceptable CPU penalty may be exacted due to the high turnover rate of entries in some implementations.

If after all of the above entries have been purged from the NODE_NAME database, no entries are available, the multicast message is discarded. Nodes associated with active virtual circuits are never purged from the node data base.

A.3.2.2.1 Error Recovery

If a connect to a SERVICE_NAME name fails, the list of SERVICE_NAMEs is searched for an alternate NODE_NAME path to the SERVICE_NAME. If one is found, a connection is attempted. This continues until a connection succeeds or until all possible NODE_NAME paths to the selected SERVICE_NAME have failed.

If 5 times the HOST_MULTICAST_TIMER seconds elapse in the terminal server, and a terminal server has not received the multicast datagram corresponding to a list entry, the entry status field is set to unknown.

A.4 Advertising Through Solicitation and Response Messages

While preserving the LAT V5.0 directory service, the LAT V5.1 architecture defines another method for a node to collect information it needs to initiate a connection to other nodes. Before attempting to establish a connection, a node may request information needed for establishing a connection using a Solicit information message. A soliciting node may multicast this message or send it physically addressed to a specific node (if this node is known as a provider of a required service). A Solicit information message can solicit information about all services, one specific service, or node information using parameters in the Solicit information message. See the section entitled "LAT Messages."

All nodes (or one specified by a name or address) that satisfy access control requirements (see "Service Access Control") and are able to process a Solicit information message, reply with a Response information message physically addressed to a soliciting node. The solicited node may respond with a Response information message that includes the following information:

- the node and all offered services
- one explicitly-requested service
- the node only (Ethernet address and some node parameters)

The soliciting node processes incoming messages and collects enough information to initiate a connection.

A node can use both methods of acquiring information about nodes and services: listening to multicasted Service announcement messages and using the Solicit information message. There is no coordination mechanism between Service announcement messages and Response information messages that would allow such a node to support a unified local cache using information from both messages. Such a node must support two separate local caches of information and update both caches separately.

More than one solicit message request may be outstanding on a soliciting node. A solicitation identifier is provided to correlate Solicit information requests with Response messages. A soliciting node inserts this identifier into the Solicit message and all replying nodes insert the same identifier into the Response message. So, all Response messages that are sent in response to this Solicit message will have the same solicit identifier.

A.4.1 A Node Operating In Slave Mode

Operations performed by a slave node may be defined as:

- The node does not listen to multicast messages.
- The node may multicast Service announcement messages containing information about services offered by that node. Multicasting is based on the multicast timer.
- The node may multicast (or send physically addressed) Solicit information messages whenever the node needs to collect information about master nodes and the services they offer.
- The node may receive and process Response information messages sent by master nodes in response to a Solicit information message.
- The implementation chooses an algorithm for processing Response information messages and methods for caching/updating a local database.

Table A–2: A Node Operating in Slave Mode

State	Event	Action	State
Halted or Soliciting	Multicast timer expired	Multicast Service announcement message	Return to existing state
Halted	solicit information: - destination node address unknown	Multicast Solicit information message	Soliciting multiple nodes
	solicit information: - destination node address is known	Send addressed Solicit information message	Soliciting specific node
	response message rec.	no action	halted
Soliciting specific node	Response time-out	Resolicit	Soliciting
	Global time-out	Abort solicitation	Halted
	Response message received	Process message	Halted
Soliciting multiple nodes	Response timeout	Resolicit	Soliciting
	Global time-out	Abort solicitation	Halted
	Response message received	Process message	Soliciting

Table A-2 assumes that each Response message is correlated to a corresponding Solicit message by means of solicitation identifiers to provide the correct state transitions.

A multicasted Solicit information message may result in some unknown number of responses. A multicasted or physically addressed Solicit information message is not guaranteed to be delivered. Therefore, a node must implement some policy for retransmitting and timing out Solicit information message requests. A response timer and a global timer can be used to retransmit a Solicit information message. The suggested method is to retransmit a Solicit information message 2 to 3 times with 1- to 2-second intervals.

A.4.2 A Node Operating In Master Mode

Operations performed by a master node can be defined as:

- The node may listen to multicast messages and support local data cache.
- The node does not multicast service announcement messages.
- After receiving a Solicit information message, the node responds with a Response information message physically addressed to a soliciting node. A Response information message can contain node information only, specified service information, or information about all services offered by the node (see "Response Information Message Policy").

To avoid a simultaneous response by all nodes, each responding node should not make its response event-driven, but rather implement a delay that will cause the node to reply within a certain time interval. The RESPONSE_TIMER parameter in the Solicit information message and some random number must be implemented by the solicited nodes to distribute the reply within the soliciting node's retransmit time interval.

One of possible algorithms of getting a random number is presented as follows: add all bytes within the unique Ethernet address resulting in a 16-bit value; use this value to initialize a counter; use the counter to count seconds since system boot; take low-order bits of the result as a random value to respond within RESPONSE_TIMER interval.

It is the responsibility of the soliciting node to implement a resolicitation algorithm as needed.

A.4.3 Response Information Message Policy

A Solicit information message can be directed to one node using a physical address or to all nodes by using a multicast address. The soliciting node can specify in the request a destination node name, the requested service name, and the requested port name. A soliciting node can request information about a node, one specific service, or all services offered by a destination node. Using the Response message, the solicited node may respond with node information only (SRVC_COUNT is 0), node and all offered services information, or node and specified service information only. The policies that govern the request/response information flow between a soliciting node and a solicited node are presented in Table A-3. A null name means that no name is provided in the Solicit information message.

Table A–3: Response Service Announcement Policy

destination service name	destination node name	Solicit service message multicasted	Solicit service message physically addressed
null	null	Any node may respond with node info and optionally all services.	Addressed node may respond with node info an optionally all services.
null	not null	Named node must respond with node info and optionally all services.	If node name is correct, node must respond with node and optionally all services info.
not null	null	If node offers service, it must respond with node and service information.	If node offers service, it must respond with node and service info. Else returns error "node doesn't offer service".
not null	not null	If named node offers service, it must respond with node and service information. Else node returns error "node doesn't offer service".	If node name is correct and node offers service, it must respond with node service info. Else return error "node doesn't offer service".

Note that the node responds only if access control requirements are satisfied (see "Access Control").

A.5 Service Class 1 Messages

Advertisement messages can be solicited or unsolicited. The Solicit information and Service announcement messages are unsolicited. The Status and Response messages are solicited messages. Each node is responsible for allocating buffers to receive maximum length unsolicited LAT messages. Unsolicited messages contain a DATA_LINK_RCV_FRAME_SIZE field that defines the maximum size of the solicited LAT messages which can be sent to the node.

Each service class can define extensions to the messages in the main body of the document.

If the slot byte count is in conflict with a field byte count, the slot is invalid. If the slot byte count truncates an extension to the slot, the slot is valid and the extension is not supplied. If a byte counted field within a slot status field is specified as zero length, the next byte following the byte count is the first byte of the following field.

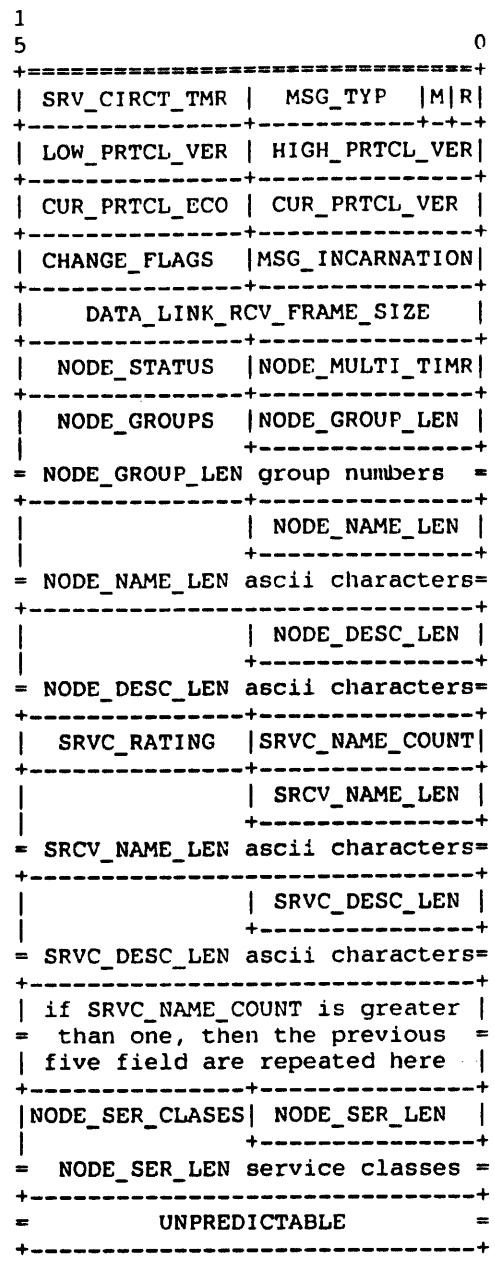
Bits are transmitted low order bit first onto the Ethernet. When fields are concatenated, the right hand field is transmitted first. Numeric fields more that 8-bits long are transmitted least significant byte first.

Fields are represented as bit streams, right to left. All fields are an integer multiple of eight bits. The symbol " = " is used to indicate fields of varying or indeterminate length.

A.5.1 Service Announcement Message

This service class defines the following additional message (this is a multicast message used in the LAT 5.0 version):

Figure A-3: Service Announcement Message



- R (1 bit) - Response requested flag. Must be zero.
- M (1 bit) - Master flag. Must be zero.
- MSG_TYP (6 bits) - fixed at 10.

- SERVER_CIRCUIT_TIMER (1 byte) - Desired value in 10 millisecond intervals. The node suggests a value in this field. The value may be ignored by the terminal server. A zero specifies no preferred value.
- HIGH_PRTCL_VER (1 byte) - Highest protocol version supported by node.
- LOW_PRTCL_VER (1 byte) - Lowest protocol version supported by node.
- CUR_PRTCL_VER (1 byte) - Protocol version of this message (current version is 5).
- CUR_PRTCL_ECO (1 byte) - ECO level of CUR_PRTCL_VER for this message (current ECO is 1).
- MSG_INC (1 byte) - Message incarnation. This multicast datagram is transmitted periodically by each node. Any time ANY field within this message changes value relative to the previous message, the MSG_INCARNATION must be incremented by one. When the node assigns this value to the first multicast message, a random value should be chosen.
- CHANGE_FLAGS (1 byte) - Each bit in this byte corresponds to a field in the multicast message that can change:
  - bit 0 - Node group codes changed
  - bit 1 - Node descriptor changed.
  - bit 2 - Service names (and/or number of names) changed
  - bit 3 - Service ratings changed.
  - bit 4 - Service descriptors changed.
  - bit 5 - Service classes changed.
  - bit 6 - unpredictable
  - bit 7 - Other parameters changed.

  If a field changes, the bit value toggles (0 -> 1 or 1 -> 0) and then remains at that value as multicast messages are transmitted until the field changes again. Only the bit(s) associated with the field(s) that have actually changed may be toggled in this way.
- DATA_LINK_RCV_FRAME_SIZE (2 bytes unsigned) - Maximum size of the LAT message that can be sent to this node. Actual length of a LAT message is DATA_LINK_RCV_FRAME_SIZE-18.

- NODE_MULTICAST_TIMER (1 byte unsigned) - The minimum rate at which the node will send multicast messages in seconds.
- NODE_STATUS (1 byte bit mask) - Node status flags byte.
  - Bit 0 - Set to 1 if the node is not accepting new sessions.
  - Bits 1 through 7 - zero on send, ignored on receive.
- NODE_GROUP_LEN (1 byte unsigned) - A byte count of the NODE_GROUP field. A value of zero is legal and indicates that the node does not offer any services.
- NODE_GROUPS (NODE_GROUP_LEN bytes) - This field is specified as a bit-mask of 256 bits. A bit set to 1 indicates the node belongs to that group. The first bit of the mask (bit 0) corresponds to group 0.
- NODE_NAME_LEN (1 byte signed) - A byte count of the NODE_NAME field. A value of zero is illegal.
- NODE_NAME (NODE_NAME_LEN bytes) - These characters are constrained as described in the section entitled "Specification of names".
- NODE_DESCRIPTOR_LEN - (1 byte unsigned) - A byte count of the NODE_DESCRIPTOR field. A value of zero indicates that no node description is available.
- NODE_DESCRIPTION (NODE_DESCRIPTION_LEN bytes) - An ASCII string of characters that describes the node.
- SERVICE_NAME_COUNT (1 byte unsigned) - This field is equal to the number of service names offered. The next five fields are repeated SERVICE_NAME_COUNT times.
- SERVICE_RATING (1 byte unsigned) - the rating of the associated service name.
- SERVICE_NAME_LEN (1 byte signed) - A byte count of the SERVICE_NAME field. A value of zero is illegal.
- SERVICE_NAME (SERVICE_NAME_LEN bytes) - These characters are constrained as described in the section entitled "Specification of names".
- SERVICE_DESCRIPTOR_LEN - (1 byte unsigned) - A byte count of the SERVICE_DESCRIPTOR field. A value of zero indicates that no service description is available.

- SERVICE_DESCRIPTION (SERVICE_DESCRIPTION_LEN bytes) - An ASCII string of characters that will help the terminal server user identify the service being offered. The node should not load control characters into this field. Terminal servers must support a minimum SERVICE_NAME length of 64 characters.

- NODE_SERVICE_LEN (1 byte unsigned) - A byte count of the NODE_SER_ CLASES field. A value of zero is illegal.

- NODE_SERVICE_CLASSES (NODE_SERVICE_LEN bytes) - A node might simultaneously support multiple service classes. A service class is coded as a byte value in the range 0 to 255. The value zero is reserved. This field is specified to make the architecture extensible. A node must specify the service classes it supports. The service classes defined at present are:

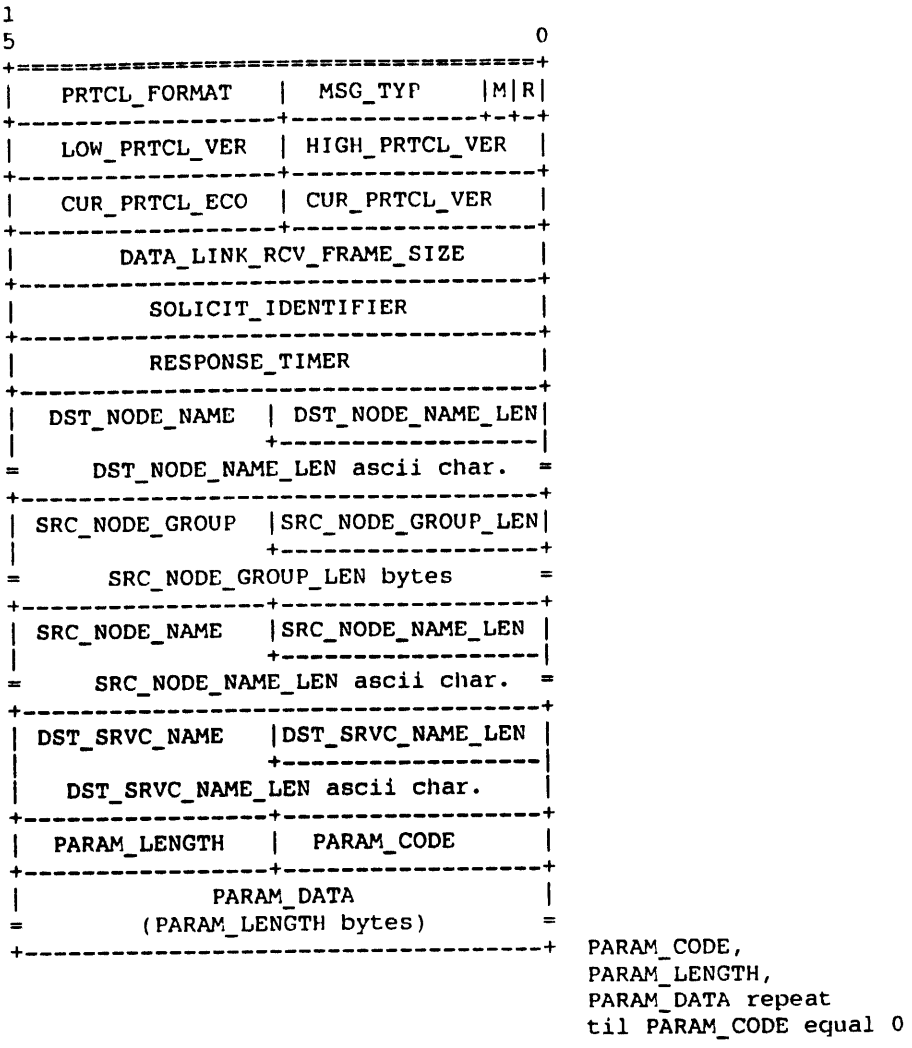
    - CLASS 1 - Interactive terminals and Application terminals

A.5.2 Solicit Information Message

The Solicit information message can be used by a node to solicit a Response information message(s) from another node(s). The Solicit information message can be multicasted or physically addressed.

Figure A-4 presents the format of the Solicit information message. Detailed descriptions of each field in the message follow.

Figure A-4: Solicit Information Message



- RM (2 bits) - Must be 0.
- MSG_TYP (6 bits) - fixed at 14.
- PRTCL_FORMAT (1 byte) - Protocol Format flag.
  - Bit 0 through 7 - Must be 0 on transmit, ignored on receive.
- HIGH_PRTCL_VER (1 byte) - Highest protocol version supported by the node.
- LOW_PRTCL_VER (1 byte) - Lowest protocol version supported by the node.

- CUR_PRTCL_VER (1 byte) - Protocol version of this message (current version is 5).

- CUR_PRTCL_ECO (1 byte) - ECO level of CUR_PRTCL_VER for this message (current ECO is 1).

- DATA_LINK_RCV_FRAME_SIZE (2 bytes unsigned) - Maximum size of the LAT message that can be sent to this node. Actual length of a LAT message is DATA_LINK_RCV_FRAME_SIZE-18.

- SOLICIT_IDENTIFIER (2 bytes unsigned) - Identifier produced by the soliciting node that uniquely identifies the Solicit information message. The soliciting node uses this identifier to correlate corresponding Solicit and Response information messages.

- RESPONSE_TIMER (2 bytes unsigned) Retransmit timer (seconds) that starts when a Command message (access) is sent. The soliciting node uses this timer to time-out waiting state for the responses. The responding node should use that value as a maximum for its local response timer.

- DST_NODE_NAME_LEN (1 byte unsigned) - length of the next field. A byte containing the length of the DST_NODE_NAME field in bytes. A value of zero is legal.

- DST_NODE_NAME (DST_NAME_LEN bytes) - Destination node name. An array of ASCII characters describing the name of the node known to be a provider of services. These characters are constrained as described in the section of the LAT Architecture Specification entitled "Specification of Names."

- SRC_NODE_GROUP_LEN (1 byte unsigned) - Node group code byte length. This byte denotes the lengths of the next field. A value of 0 is legal and indicates that node can access any services. Maximum value is 32 (256 bits).

- SRC_NODE_GROUPS (SRC__NODE_GROUP_LEN bytes) - Soliciting node group code mask. This field is specified as a bit-mask of up to 256 bits. A bit set to 1 indicates the node belongs to that group. The first bit of the mask (bit 0) corresponds to group 0. This group code mask represents Identifiers List (IDL).

- SRC_NODE_NAME_LEN (1 byte unsigned) - Length of the next field. A byte containing the length of the SRC_NODE_NAME field in bytes. A value of zero is illegal.

- SRC_NODE_NAME (SRC_NODE_NAME_LEN bytes) - Soliciting (source) node name. An array of ASCII characters describing the name of the node soliciting a response message. These characters are constrained as described in the section of the LAT Architecture Specification entitled "Specification of Names."

- DST_SRVC_NAME_LEN (1 byte unsigned) - Service name length. A byte containing the length in bytes of the DST_SRVC_NAME field. A value of zero indicates no Service name is requested.

- DST_SRVC_NAME (DST_SRVC_NAME_LEN bytes) - Requested service name. An array of ASCII characters that forms the name of the requested service. These characters are constrained as described in the section of the LAT Architecture Specification entitled "Specification of Names." This name is not constrained to be unique in the Local Area Network because the same service name can be offered by different nodes.

- PARAM_CODE (1 byte) - Parameter code. The following codes are defined:

  - Parameter code 0 - Denotes the end of a parameter list.

  - Parameter codes 1-127 - Reserved for DEC use.

  - Parameter codes 128-255 - Reserved for users.

- PARAM_LEN (1 byte) - Length of the next field in bytes.

- PARAM_DATA (PARAM_LEN bytes) - Parameter data.

A.5.3 Response Information Message

The Response information message is physically addressed to the soliciting node. The Response message must fit in the receive buffer provided by the soliciting node (see "Solicit Information Message"). The more general solution needed in the case when the Response message does not fit in the provided buffer is outside the scope of the LAT V5.1 architecture.

Figure A-5 presents the format of the Response information message. A detailed description of each field in the message follows.

Figure A-5: Response Information Message

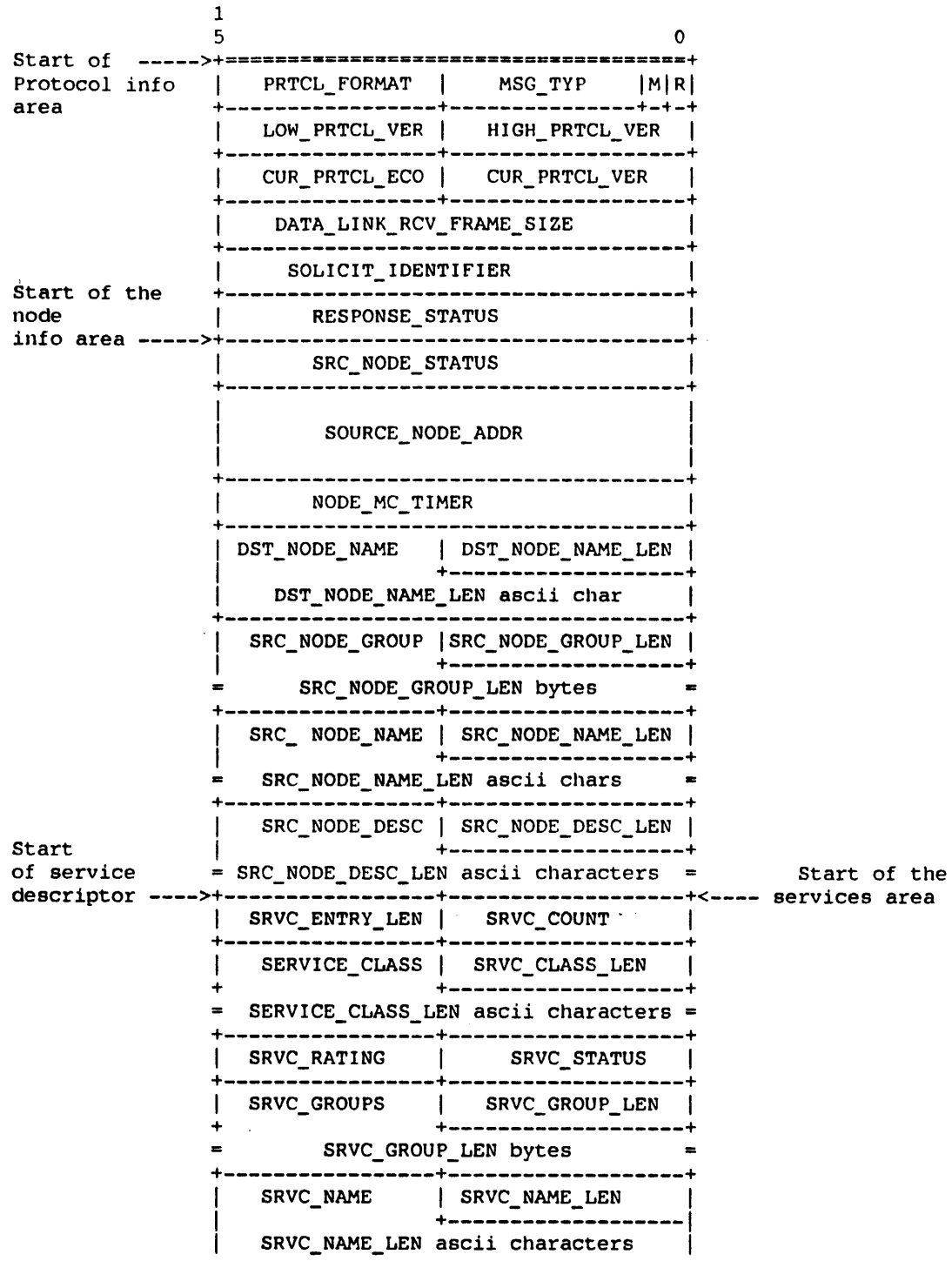
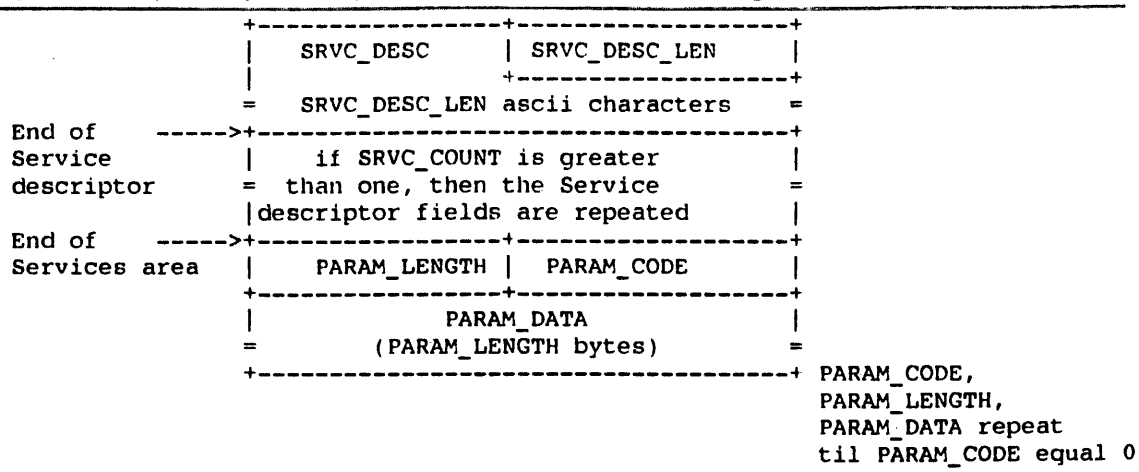


Figure A-5 Cont'd. on next page

Figure A-5(Cont.): Response Information Message



- RM (2 bits) - Must be zero.
- MSG_TYP (6 bits) - Fixed at 15.
- PRTCL_FORMAT (1 byte) - Protocol Format flag. Bits 0 - 1 define mode of operation of the node as follows:
  - 00 - node can operate in the Ethernet format only
  - 01 - node can operate in the 802 format only
  - 10 - node can operate in both - 802 and Ethernet formats
  - 11 - reserved
  - Bits 2 through 7 - Must be 0 on transmit, ignored on receive.
- HIGH_PRTCL_VER (1 byte) - Highest protocol version supported by the node.
- LOW_PRTCL_VER (1 byte) - Lowest protocol version supported by the node.
- CUR_PRTCL_VER (1 byte) - Protocol version of this message (current version is 5).
- CUR_PRTCL_ECO (1 byte) - ECO level of CUR_PRTCL_VER for this message (current ECO is 1).
- DATA_LINK_RCV_FRAME_SIZE (2 bytes unsigned) - This field must be zero on send; ignored on receive.

- SOLICIT_IDENTIFIER (2 bytes unsigned) - This value is equal to the value of the SOLICIT_IDENTIFIER field of the received Solicit information message that is being replied to by this Response information message.
- RESPONSE_STATUS (2 bytes unsigned) - Response status. Bit mask representing status of the Response information message. Meaning of bits (when set):
  - Bit 0 - reserved
  - bit 1 - Node does not offer requested service.
  - bits 2-15 - Must be zero.

The next field marks the start of the node information area.

- SRC_NODE_STATUS (2 byte bit mask) - Responding node status flag word represented by a bit mask as follows:
  - Bit 0 set - the node is disabled (node is not accepting new connections to its services).
  - bit 1 set - Start message can be sent by the subject node to the object node that issued the Response information message.
  - bit 2 set - Command message can be send by the Subject node to the Object node that issued the Response information message.
  - Remaining bits - must be 0 on send, ignored on receive.

  Bits 1 and 2 in the SRC_NODE_STATUS field define functional capabilities of the solicited node. Using those two bits in the NODE_STATUS byte of the Response message, a responding node reports to the solicitor what messages the subject can use to initiate a connection to the object. Table A-4 explains the meaning of those bits and combinations.

Table A-4: SRC_NODE_STATUS Bit Combinations

Bits 2 1	Meaning
0 0	Object does not accept either Command or Start message. No connection can be made to this object (5.0 server is an example of a such node).
0 1	Subject can send a Start message. Object will respond with Start/Reject message only. (5.0 host is an example of a such node).

Table A–4 (Cont.): SRC_NODE_STATUS Bit Combinations

Bits 2	1	Meaning
1	0	Subject can send Command message. Object can respond with Start/Reject or Status messages. (5.1 server that offers services is an example).
1	1	Subject can send both Start and Command messages. Object can respond with both Start and Status messages. By advertising ability to receive a Start message, object announces itself to be at least a slave. (5.1 slave which provides queuing to the services and symmetric node are examples of such nodes).

- SOURCE_NODE_ADDR (6 byte field) - transmitting node must fill this field with the value equal to the data link source address of the node. Receiving node must reference this field and must ignore the actual source address of the message.

- NODE_MC_TIMER (2 bytes unsigned) - Slave node multicast timer (the maximum time between transmitted Service messages in seconds). The value must be in the range 1 to 3600 seconds.

- DST_NODE_NAME_LEN (1 byte unsigned) - Length of the next field. A byte containing the length of the DST_NODE_NAME field in bytes. A value of zero is legal.

- DST_NODE_NAME (DST_NAME_LEN bytes) - Destination node name. An array of ASCII characters describing the name of the node to which the Response message is directed. These characters are constrained as described in the section of the LAT Architecture Specification entitled "Specification of Names."

- SRC_NODE_GROUP_LEN (1 byte unsigned) - Node group code byte length. This byte denotes the length of the next field. A value of 0 is legal and indicates that the node is offering no services. The maximum value is 32 (256 bits).

- SRC_NODE_GROUPS (SRC_NODE_GROUP_LEN bytes) - Node group code mask. This field is specified as a bit-mask of up to 256 bits. A bit set to 1 indicates the node belongs to that group. The first bit of the mask (bit 0) corresponds to group 0. This group code mask represents an Access Control List (ACL).

- SRC_NODE_NAME_LEN (1 byte unsigned) - Node name length. A byte count of the following field. A value of zero is illegal.

- SRC_NODE_NAME (SRC_NODE_NAME_LEN bytes) - Node name. An ASCII string of characters that contains the name of the node for which the information applies. These characters are constrained as described in the section of the LAT Architecture Specification entitled "Specification of Names." This name must be unique to the Local Area Network.

- SRC_NODE_DESC_LEN (1 byte unsigned) - Node description length. A byte count of the following field. A value of zero indicates that no node description is available.

- SRC_NODE_DESC (SRC_NODE_DESC_LEN bytes) - Node description. An ASCII string of characters representing the textual description of the node.

The next field marks the start of the Services area.

- SRVC_COUNT (1 byte unsigned) - Service count. This is the total number of service entries included in the message.

The next field marks the start of the Service entry.

- SRVC_ENTRY_LEN (1 byte unsigned) - Number of bytes in this service entry. It is used to speed the search through the entries presented in the list.

- SRVC_CLASS_LEN (1 byte) - length of the following field in bytes. A value of zero is legal and means that service belongs to class 1.

- SERVICE_CLASS (SRVC_CLASS_LEN bytes) - class of the described service (service can belong to more then 1 class). Equal 1 for the interactive and application terminals.

- SRVC_STATUS (1 byte unsigned) - Service status. This field is specified as a bit mask of 8 bits. The bits are defined as follows:

  - Bit 0 set - Service is enabled.

  - Bit 1 set - Service supports queuing (see "Service Sharing").

  - Remaining bits - must be 0 on send, ignored on receive.

- SRVC_RATING (1 byte unsigned) - The rating of the associated Service. This value changes dynamically depending upon type of service, system resources, number of users, etc.

- SRVC_GROUP_LEN (1 byte unsigned) - Service group code length. A byte count of the SRVC_GROUP field. A value of 0 is legal and indicates that the service group codes are not available (use node group codes). The maximum value is 32 (-> 256 bits).

- SRVC_GROUPS (SRVC_GROUP_LEN bytes) - Service group codes. This field is specified as a bit-mask of 256 bits. A bit set to 1 indicates the service belongs to that group. The first bit of the mask (bit 0) corresponds to group 0. This group code mask represents an Access Control List (ACL).

- SRVC_NAME_LEN (1 byte unsigned) - Service name length. A byte containing the length in bytes of the SRVC_NAME field. A value of zero is illegal.

- SRVC_NAME (SRVC_NAME_LEN bytes) - Service name. An array of ASCII characters that forms the name of the service. These characters are constrained as described in the section of the LAT Architecture Specification entitled "Specification of Names." This name is not constrained to be unique in the Local Area Network.

- SRVC_DESC_LEN - (1 byte unsigned) - Service description length. A byte count of the SRVC_DESC field. A value of zero indicates that no description is available.

- SRVC_DESCRIPTION (SRVC_DESC_LEN bytes) - Service description. An ASCII string of characters that describes the service. For an application terminal service, this is typically the device location.

  The previous field marks the end of the Service description entry.

- PARAM_CODE (1 byte) - Parameter code. The following codes are defined:

  - Parameter code 0 - Denotes the end of the parameter list.

  - Parameter codes 1-127 - Reserved for DEC.

  - Parameter codes 127 - 255 - Reserved for users.

- PARAM_LEN (1 byte) - Length of the next field in bytes.

- PARAM_DATA (PARAM_LEN bytes) - Parameter data.

A.6 Service Class 1 Slot Format Extensions

To accommodate functionality and features of the Class 1 service some slots are extended by this service class.

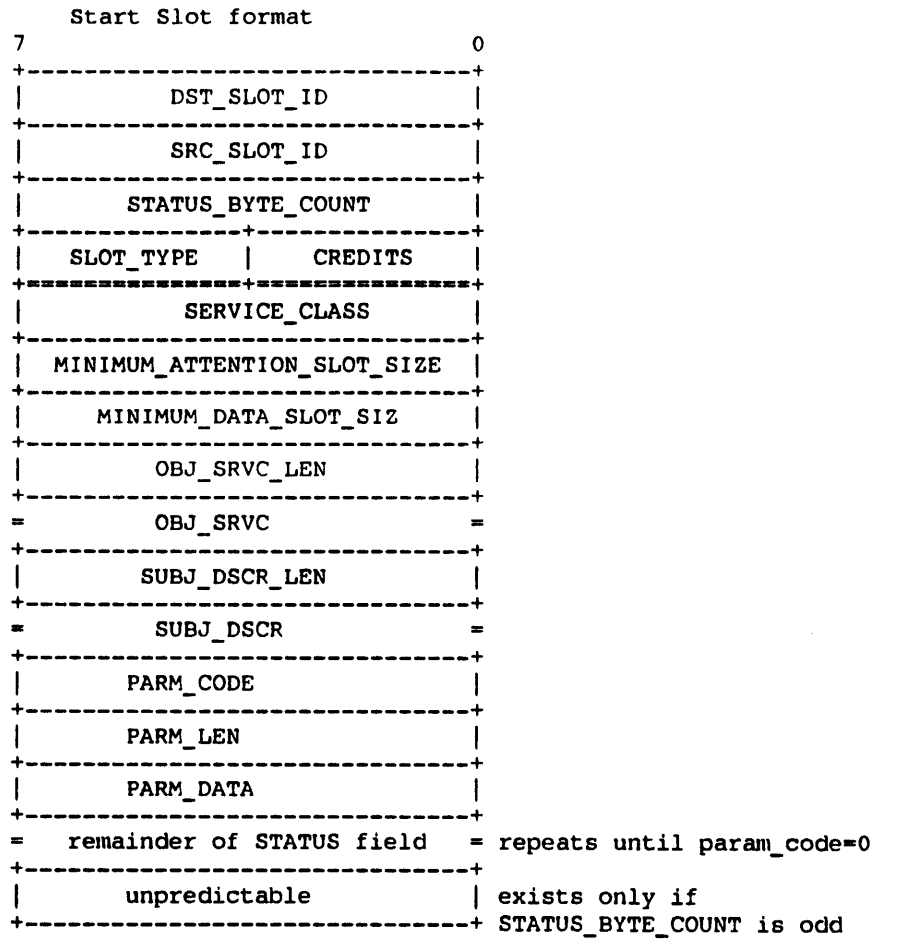
A.6.1 Start Slot Status Field

Start slot status field is extended by this service class.

As was mentioned before, a node using a Solicit service message to request Response service announcements can use different methods to process responses (limited caching, filtering, etc.). Other nodes can use Start slots to initiate connections. To provide naming conventions as specified by the LAT architecture, the Start slot contains destination and source service and port names.

To preserve compatibility with the LAT 5.0 implementations and provide easy ECOing, new fields have been incorporated into the Start slot within the parameter field.

Figure A–6: Start Slot Format



- DST_SLOT_ID (1 byte) - A reference to a slot block.

- SRC_SLOT_ID (1 byte) - A reference to a slot block.
- STATUS_BYTE_COUNT (1 byte) - An unsigned integer count of the length of the STATUS field.
- CREDITS (4 bits) - A 4-bit integer equal to the number of credits being transferred.
- SLOT_TYPE (4 bits) - The value 9 (1001).
- SERVICE_CLASS (1 byte) - The value 1 for application and interactive terminals.
- MINIMUM_ATTENTION_SLOT_SIZE (1 byte) - The minimum slot size queued to receive Attention slot data (not including the slot header). The system receiving this message must limit transmitted Attention slots to this size. A value of zero indicates Attention slots are not supported.
- MINIMUM_DATA_SLOT_SIZE (1 byte) - The minimum slot size queued to receive Data_a and Data_b slots (not including the slot header). The system receiving this message must limit transmitted Data_a and Data_b slots to this size. A value of 0 is illegal.
- OBJ_SRVC_LEN (1 byte unsigned) - The byte count of the next field. A value of zero indicates that no service name is provided.
- OBJ_SRVC (OBJ_SRVC_LEN) - When a Start slot is sent by the initiator, this field specifies the destination service name. When a Start slot is sent by a responder, this field is the result of the destination service name translation process.
- SUBJ_DSCR_LEN (1 byte unsigned) - The byte count of the next field. A value of zero indicates that no textual description is provided.
- SUBJ_DSCR (SUBJ_DSCR_LEN bytes) - When sent by an initiator, this field specifies the subject textual description. When sent by a responder,this field must be 0 on send and ignored on receive.
- The following Start Slot parameters are defined:
  - Parameter code 0 is reserved.
  - Parameter code 1(2 bytes) - Flag word; bits when set are:
    - Bit 0 - If set indicates a dialup line. If cleared indicates a local line. Intendent to be settable by a server manager to indicate a dialup line.

- Bit 1 - if set this line does not automatically initiate a login sequence in the Start slot.
- Bits 4-15 - Zero on send ignored on receive.

- Parameter code 2 - identifier of the particular entry in the queue (2 bytes unsigned) - This field contains the unique identifier assigned to the queue entry by the node to which the Start is directed. This value correlates the connection request with the service queue. If the session was not solicited, this parameter should not be specified.
- Parameter code 3 (2 bytes) - reserved (zero on send, ignored on receive).
- Parameter code 4 - OBJ_PORT_NAME (string of bytes PARM_LEN long) - Destination node port name. This parameter is used to designate a particular port on a destination node.
- Parameter code 5 - SUBJ_PORT_NAME (string of bytes PARM_LEN long) - Source node port name. This parameter is used to designate a particular port on a source node.
- Parameter code 6 - SUBJ_GROUP_CODES (string of bytes PARM_LEN long) - Source service group codes. This field is specified as a bit-mask up to 256 bits. A bit set to 1 indicates the subject belongs to that group. The first bit of the mask (bit 0) corresponds to group 0. This group code mask represents an Identifier List (IDL). It must not be present in the Start (response) slot.
- Parameter code 7 - OBJ_SRVC_PASS (string of ASCII characters PARM_LEN long) - Service password. This parameter is used to pass a per-service password to the object node (for implementations that use service passwords).
- Parameter codes 8-127 - Reserved for DEC.
- Parameter codes 128-255 - Reserved for users.

Note that no queue depth indicator is returned in the Start Slot since the establishment of the connection means that the service is 'online' to the solicitor.

A.6.2 Attention Slot Status Field

The Attention slot is extended by this service class to include 1 byte of control flags, and within the byte a single "abort" flag. It's purpose is to discard all buffered data remaining to be delivered to the user. The slot can be sent by either the host or the terminals server.

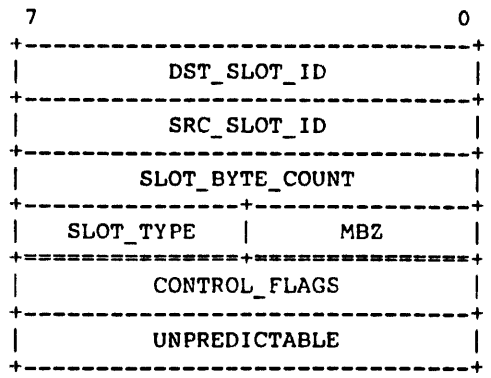
Host implementation is optional for both transmission and reception of the slot.

The terminal server must process this slot if it is received, but transmission of this slot is optional.

Note that this slot is not flow controlled.

The minimum Attention slot size is 1 byte. The format of the Attention slot is:

Figure A-7: Attention Slot Format



- DST_SLOT_ID - a handle on the remote slot block
- SRC_SLOT_ID - a handle on the local slot block
- SLOT_BYTE_COUNT - an unsigned integer count of the length of the SLOT_DATA field - the value 1.
- MBZ (4 bits) - must be zero
- SLOT_TYPE (4 bits) - the value 11.
- CONTROL_FLAGS (8 bits) :
  - (bit 0 through bit 4) - Unpredictable.
  - (bit 5) - Abort. Causes buffered output data pipe to be flushed of all data. Credits are returned as if the data had been normally delivered.
  - (bit 6 and bit 7) - Unpredictable.

A.6.3 Data_b Slot Extension

The data_b slot is extended by this service class to provide port control and information, session control and data stream information functions.

The flow control discussed in this chapter concerns only the communication between the DTE and the session and not the LAT session flow control which is credit based as described in the previous chapters.

The LAT V5.1 architecture supports only XON/XOFF flow control (DEC STD 110). Other flow control mechanisms are outside the scope of this architecture.

A.6.3.1 Information Exchange Using Data_b Slots

The LAT architecture defines two types of a data_b slots - "Set" and "Report". By using different types of data_b slots each of the sessions may coordinate the setting and display of remote port characteristics and data transparency mode. This design does not require a specific implementation. Actual implementation of the set/report data_b slots is a product specific issue. Products can choose to implement or not to implement this functionality.

Information about physical port characteristics, the status of the data stream and the setting of the data transparency mode (described below) is communicated between connecting nodes using Data_b slots. The following information is included in Data_b slots:

Port	Session	Data Stream
Receive Speed Transmit Speed Parity Type Data Length Input Flow Control Output Flow Control Bell on Discard	Transparency Mode: - none - passall - pasthru	Break Condition Data Error Detected

Each end of the communication session can be conveniently described as having three databases. One database describes the characteristics of the physical port on the local end (if one exists), one database is the image of physical characteristics of the port on the remote end of the connection (if one is needed), and one database describes the current session characteristics.

"Set" and "report" data_b slot types are represented by bits 5 and 6 in the CONTROL_FLAGS byte of the Data_b slot as follows:

- bit 5 and bit 6 are both cleared - V5.0 compatibility (in most cases, the data_b slot of this type will be processed as "set" if the physical port is present).
- bit 5 is set, bit 6 is cleared - "Set" type of data_b slot.
- bit 5 is cleared, bit 6 is set - "Report" type of data_b slot.
- both bits set - this combination must not be specified on transmit, and a data_b slot with this combination must be ignored on receive.

The 'report' data_b slot contains all the information about the local port characteristics and the transparency mode setting regardless of what information has been changed. A node sends a 'report' data_b slot when information in the local database has been changed. The 'set' data_b slot contains information on only those parameters that are being changed. It is sent by the node which is attempting to modify remote physical port characteristics or the transparency mode setting for a session.

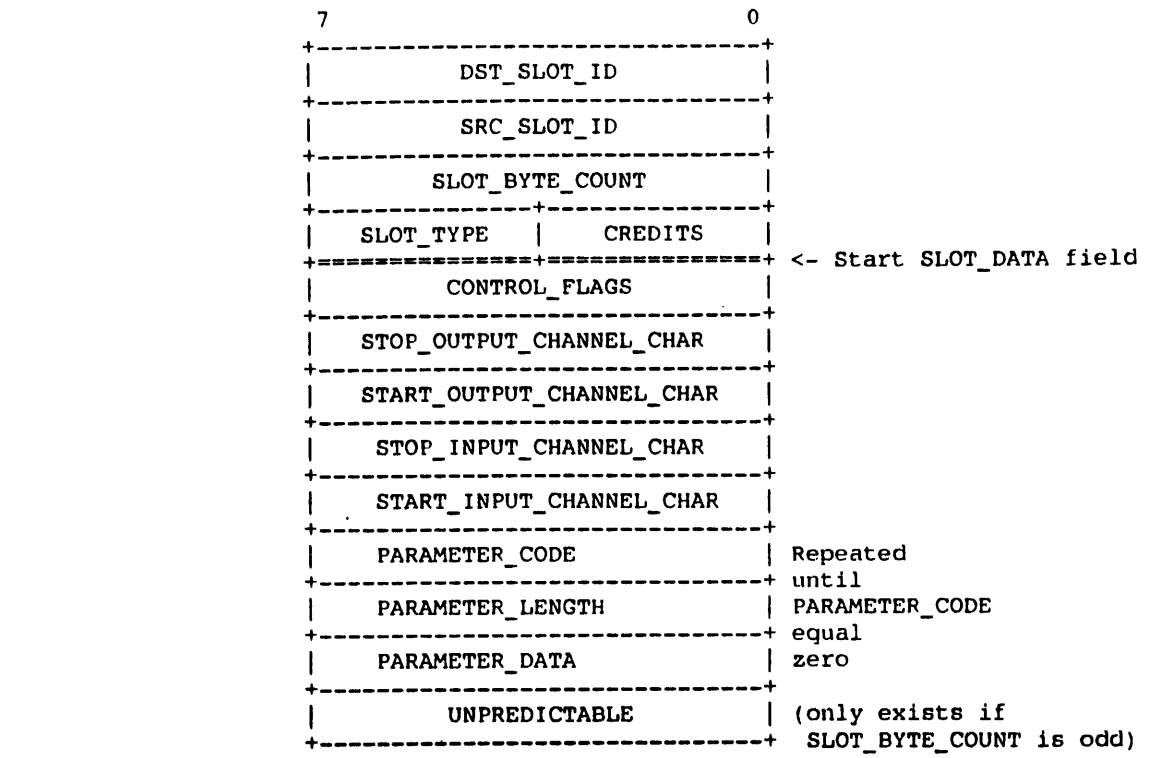
A.6.3.2 Data_b Slot Format

The format of the data_b slot is shown in Figure 6-1.

Figure A–8: Data_b Slot Format

Figure A–8 Cont'd. on next page

Figure A-8(Cont.): Data_b Slot Format



- DST_SLOT_ID (1 byte) - A handle on the remote slot block.
- SRC_SLOT_ID (1 byte) - A handle on the local slot block.
- SLOT_BYTE_COUNT (1 byte) - An unsigned integer count of the length of the SLOT_DATA field.
- CREDITS (4 bits) - A positive integer equal to the number of credits being extended.
- SLOT_TYPE (4 bits) - The value 10.
- CONTROL_FLAGS (8 bits) - Control flags. The bit settings are as follows.
  - bit 0 - Enable usage of input flow control characters. This bit changes the meaning of STOP_INPUT_CHANNEL_CHAR and START_INPUT_CHANNEL_CHAR in the terminal output stream. If the terminal server is about to overflow the terminal input stream buffer, it should insert the STOP_INPUT_CHANNEL_CHAR into the terminal output stream. When sufficient input buffering is again available, the START_INPUT_CHANNEL_CHAR must be inserted into the terminal output stream.

- bit 1 - Disable recognition of input flow control characters. No characters are generated by the terminal server and inserted into the output stream to control the input data flow (e.g., when the server input buffer is overflowing).

- bit 2 - Enable recognition of output flow control characters. This bit changes the meaning of STOP_OUTPUT_CHANNEL_CHAR and START_OUTPUT_CHANNEL_CHAR in the terminal input stream. Upon detecting one of these characters, the terminal server should disable/enable the terminal output stream as indicated. The STOP_OUTPUT_CHANNEL_CHAR and START_OUPUT_CHANNEL_CHAR flow control characters are discarded by the terminal server from the input stream (i.e., they are not passed to the host).

- bit 3 - Disable recognition of output flow control characters. All characters in the terminal input stream are passed directly through to the host without interpretation by the terminal server.

- bit 4 - Break condition detected. Parameter code 6 in the parameter list defines a long or short break signal. Parameter code 6 is used only if long/short break signal can be distinguished.

- bit 5 - Set port characteristics The characteristics are represented by the specific parameter codes 1 through 5 in the parameter list.

- bit 6 - Report port characteristics. the characteristics are represented by the specific parameter codes 1 through 5 in the parameter list.

- bit 7 - must be 0 on send; ignored on receive.

Pairs of bits that cannot be simultaneously set in the same data_b slot are: bits 0 and 1, bits 2 and 3, bits 5 and 6.

- STOP_OUTPUT_CHANNEL_CHAR (1 byte) - The value assigned to stop the terminal output stream if output flow control characters are enabled. The value assigned is always control-S. When the terminal server detects this character in the input stream, it immediately stops any output to the terminal. The terminal server discards this flow control character from the input stream (i.e., it does not pass it to the host). The terminal server interprets this character as a flow control character only if bit 2 is set in the CONTROL_FLAGS byte.

- START_OUTPUT_CHANNEL_CHAR (1 byte) - The value assigned to start the output channel if output flow control characters are enabled. The value assigned is always control-Q. When the terminal server detects this character in the input stream, it resumes any output that was previously stopped because the STOP_OUTPUT_CHANNEL_CHAR character was seen in the input stream. The terminal server discards this flow control character from

the input stream (i.e., it does not pass it to the host). The terminal server interprets this character as a flow control character only if bit 2 is set in the CONTROL_FLAGS byte.

- STOP_INPUT_CHANNEL_CHAR (1 byte) - The value assigned to stop the terminal input channel if input flow control characters are enabled. The value assigned is always control-S. The terminal server outputs this character when its local buffer for the input data stream begins to overflow. The terminal server uses this character as a flow control character only if bit 0 is set in the CONTROL_FLAGS byte.

- START_INPUT_CHANNEL_CHAR (1 byte) - The value assigned to start the terminal input channel if input flow control characters are enabled. The value assigned is always control-Q. The terminal server outputs this character when sufficient local buffering for the input stream exists to resume input previously suspended by the STOP_INPUT_CHANNEL_CHAR. The terminal server uses this character as a flow control character only if bit 0 is set in the CONTROL_FLAGS byte.

- PARAMETER_CODE (1 byte) - The following codes are defined:

  1. code 0 - Denotes the end of the parameter list.

  2. code 1 (1 byte) - the parity and the frame size.

     - bits 0-3 - Bits per character (not counting parity bits).

     - bit 4 - Parity enabled if set; parity disabled if cleared.

     - bits 5-6 - Type of parity (00 = space, 01 = odd, 10 = even, 11 = mark).

     - bit 7 - Reserved; must be 0 on send; ignored on receive.

  3. code 2 - INPUT_SPEED (2 bytes unsigned) - The approximate input data rate of the service in bits per second. This field only has meaning for services that are application terminals. A value of zero indicates that the speed is unknown. An octal value of 177777 is taken to mean that the speed is in excess of 64k bits.

  4. code 3 - OUTPUT_SPEED (2 bytes unsigned) - The approximate output data rate of the service in bits per second. This field only has meaning for services that are application terminals. A value of zero indicates that the speed is unknown. An octal value of 177777 is taken to mean that the speed is in excess of 64k bits.

  5. code 4 (1 byte) - User preference feature, which has a value of:

     - 0 - Disable bell-on-discard.

- 1 - Enable bell-on-discard.

6. code 5 (1 byte) - Transparency mode, which has a value of:
   - 0 - Normal mode (passall and pasthru are disabled).
   - 1 - Enable passall mode.
   - 2 - Enable pasthru mode.
7. code 6 (2 bytes) - Status. The first byte is a status code as follows:
   - 0 - Unknown error.
   - 1 - Short break detected.
   - 2 - Long break detected.
   - 3 - Framing error.
   - 4 - Data overrun.
   - 5 - Parity error.

   For status codes 3, 4, and 5, the second byte is an image of the received byte.

- codes 7-127 - Reserved for DEC use.
- codes 128-255 - Reserved for users.

Note, that parameter codes 1-5 can be present in both - "set" and "report" data_b slot types. Parameter code 6 can be present only in the "report" data_b slot type.

A.6.3.3 Guidelines And Recommendations For Data_b Slot Processing

The following section does not present a requirement for implementing data_b slots processing in products. It represents some guidelines and ideas of how to deal with flow control and transparency mode using data_b slots. Each implementation can use data_b slots in designing the behavior most appropriate for the particular product. The guidelines and implementation examples given below clarify possible usage of data_b slots allowing cooperating products to distribute the control of the physical port and session characteristics, change transparency mode and update the displayed port characteristics.

A.6.3.3.1 Port Characteristics

There are a number of characteristics that are considered to be attributes of the port and not of the session:

- Receive Speed
- Transmit Speed
- Parity Type
- Data Length
- Input Flow Control Method
- Output Flow Control Method
- Bell on Discard

These characteristics define the manner in which the port interacts with the equipment at the other end of the line. Port characteristics are visible on the server side as well as on the host. After the session has been established, the current port characteristics must be reported to the connected node. Also, any time any of the port characteristics are changed, any connected session must be notified of the change. Architecturally port characteristics are settable not only from the server, but also from the host over the currently active session by using mechanisms defined in the architecture.

Following presumtions allow to organize flow control, port characteristics setting and data transfer:

- Flow control concerns only the communication between the terminal and the server port, and is restricted to an enabling/disabling XON/XOF. Recognition of the switch characters (XON/XOFF), the BREAK key, and flow control characters, and insertion of those characters in a data stream is dealt with by transparency mode (see below).
- Flow control is a port rather than a session characteristic. Characteristics of the port, once set, will stay that way until another SET CHARACTERISTICS command is issued from a local terminal or from a remote node. That is, flow control characters set within one session will not be changed or restored when the user switches to another session (including switching to a local session). Setting flow control from a local terminal has the same effect as a flow control command received from a remote node.

A.6.3.3.2 Session Management

The setting of switch characters, including the BREAK key, is part of the session management context. Transparency mode is a characteristic of the session. The transparency modes are:

- none - No transparency mode. The BREAK key may be set to be LOCAL (i.e., switch character), REMOTE (i.e., signaled to the host), or ignored depending on the product and user requirements.
- passall - All characters including XON/XOF, forward and local switch characters, and BREAK are transferred through the data stream. Intent is to provide uninterraptable channel for binary data communication purposes between computers. Conformance with this mode is a product specific issue.
- pasthru - XON/XOF are still flow control characters, but all other characters, including BREAK and forward and local switch characters, are transferred through the data stream.

Data transparency is provided on three levels:

- Disabling filtering of the switch characters in the input stream and insertion of the switch characters in the output stream.
- Disabling recognition and interpretation of the in-band flow control characters.
- Setting the BREAK key in REMOTE mode.

The duration of the data transparency setting will be from the time transparency mode is enabled until either transparency mode is disabled or the session is interrupted. Possible ways of dealing with an interrupted session include session termination, error indication, and the possibility of resuming a session. The control and visibility of the data transparency is the same as for port characteristics. Other methods of interrupting a session in transparency mode are product or user requirements issues.

A.6.3.3.3 Data_b Slot Processing

The local database represents characteristics of the local port and is shared among all connected sessions. Therefore when port characteristics change, all connected sessions must be notified about the change. The data transparency mode setting and the remote database (i.e. characteristics of the remote port) are part of the session context and therefore applicable only to a particular session.

Sending of a data_b slots:

- A node sends a 'Report' data_b slot to inform the remote node of it's current port and session characteristics when the port characteristics in the local database are modified by a user through a local command. All connected sessions should be notified about the change.

- When a characteristic belonging to the remote database is modified, a "set" type data_b slot may be sent in order to modify port characteristics on the remote node (result of this operation depends upon implementation on the remote node).

- Each node must send a 'report' type data_b slot after a session is started to inform the partner about the current port characteristics and desired data transparency mode.

- When the session database is modified (by a user command), both ends of the connection are affected, and the local node should send a "Set" type data_b slot. When session characteristics are specified in a "Set" data_b slot, the receiving node interprets the request to mean that the requesting has changed the characteristic and the object should also change it. The sending node should update the remote database if one exists. The receiving node should change session characteristics when it receives this slot if it has a physical port. However, nodes without physical ports may not be able to change session characteristics, and thus this operation is not guaranteed.

- A node sending a 'Set' type data_b slot must not go to a waiting state for the return of the report slot. All processing of the information is done whenever the 'Report' slot is actually received. Since slots are only processed for current sessions, a 'Set' slot sent to a dormant (non-current) session may not be processed for some time or the partner node may not the implement functionality to change its local database.

Receiving of a data_b slots:

- The node which receives a 'set' data_b slot should issue a command the local physical port (if applicable) and update the local database. In order to insure proper operation 'report' data_b slot must be sent by the node when it receives and reacts to a 'set' data_b slot.

- If a "report" data_b slot is received, the node updates the remote database (if applicable). Though remote database was updated, node must not send a "set" data_b slot to avoid a "set"-"report" loop.

A.6.3.3.4 Implementation Examples

- A "Host" node would not have a local database (local port does not exist) but does have a remote database (image of the port on the server). Executing "set terminal" commands node would modify the remote database and would send a 'Set' data_b slot. Terminal server on the receiving end end could elect to change or not to change the port characteristic, depending on the desired results of the server. The server would respond with a 'Report' data_b slot. When a "report" data_b slot is received, the host will modify its remote database. A "show terminal" commands display data from the remote set.

- A server has a local database (real physical port) and does not have a remote database. Local "Set" and "Show" commands operate on the local database. After a local "Set" command, the server updates it's local database and should send a 'Report' DATA_B slot to all sessions associated with that port (therefore making new characteristics available to all connected sessions).

- When both a local and a remote database exist (port-to-port connection), a different type of SET/SHOW command can be implemented to control characteristics of communicating physical ports and their image in a remote database.

- A server user issues a local "Set Session" command to change the data transparency mode of that session. The server does not know if it is communicating with a "host" system or with a "reverse server". The server issues a 'Set' DATA_B slot (for that session only). The receiving node updates it's session characteristic database and sends a 'Report' DATA_B slot in return. Note that in the case of a "reverse server", this action is essential to allow transparent data transfer operate properly.

- A server user has the BREAK characteristic set to REMOTE and, while in host mode, enters a break character. The server sends a 'Report' type DATA_B slot (to the current session only) with the break detected bit set.

- A session is created on a server. The server sends a 'Report' type DATA_B slot after receiving the slave start slot. This 'Report' slot contains complete information on all port and session characteristics.

- A user on a server issues a local "Set session passall" command to change the data transparency mode of a session connected through a reverse server. The local server is using LAT V5.1 and the remote server is using LAT V5.0. The DATA_B slot is sent with the disable input and output flow control bits set.

B

Compatibility and Implementation

Following chapter discusses some of the compatibility and implementation issues between products implementing LAT 5.0 and LAT 5.1 versions of the architecture.

B.1 Implementation Issues

A number of different services can be presented using messages defined by the Service Class 1 architecture. Some of the implementation issues are discussed below.

B.1.1 Possible Implementations of the LAT V5.1 architecture

LAT architecture is build on the principle of "modularity". That means product implementators can choose what features of the LAT 5.1 architecture to implement to build a product with desirable characteristics. Three major architectural functions introduced by LAT 5.1 architecture are:

- information solicitation/response
- connection solicitation
- queuing

Any of these features can be implemented independently of others in each particular product (for example host node can provide initiate connection to the terminal server services without supporting information solicitation, or terminal server can avoid implementing queuing, etc). Different products can choose an implementation that combines some of those features to achieve required product functionality:

- connection solicitation will be always supported by the LAT 5.1 products in order to provide host-driven connection to services on terminal servers.

- if host node does not implement information solicitation/response algorithm, then Ethernet addresses must be manually introduced into the system, and this information must be constantly updated keeping the database in order.
- terminal server that does not support information response will be invisible to the soliciting host nodes, i.e. host nodes should be provided with this information in some other fashion (manually);
- if the host node does not implement connection solicitation algorithm then node can't access services on the servers (LAT 5.0);
- terminal server that does not support queuing will reject connection to the busy resource and queuing node will have to repeat connection request in order to get connection.
- queuing also can be implemented in a fashion where hosts can queue requests to the provided services and servers can use connection solicitation mechanism to queue requests. That would allow the host node also queue requests to the offered services.

Table B-1 presents some possible implementations of the LAT V5.1/5.0 products and their relations in terms of listening and reacting upon LAT messages on the Ethernet.

Table B–1: LAT V5.1 Implementations and LAT Messages

	Listens to multicast msgs	Sends multicast msgs	Solicits & listens to response	Listens to solic. & responds	Solicits connect	Responds -status, provides queues	Initiates start	Listens to start & response start
1	-	+	-	-	-	-	-	+
2	+	-	-	-	-	-	+	-
3	-	+	+	-	+	-	-	+
4	-	+	+	-	+	+	-	+
5	+	-	-	+	-	+	+	-
6	+	-	-	+	+	+	+	-
7	+	+	-	-	-	-	+	+
8	+	+	+	+	+	+	+	+

In the table above sign "+" means "feature is implemented" and sign "-" means "feature is not implemented". Short description of the products described in the table is given below:

- 1 - 5.0 host
- 2 - 5.0 server
- 3 - 5.1 host without queued services
- 4 - 5.1 host with queued services
- 5 - 5.1 server without queuing to hosts
- 6 - 5.1 server with queuing to hosts
- 7 - 5.0 symmetric server (slave and master within one node)
- 8 - 5.1 symmetric server (slave and master within one node)

B.1.2 Local Data Base

Terminal servers in the LAT 5.0 architecture may support the full data base of nodes and advertised services. It is recommended that LAT products support full data base. If LAT 5.1-based product is not able to support full data base for the lack of resources, advertising mechanism architectured into the LAT 5.1 version allows this node to use Solicitation/Response messages to acquire information needed for the connection establishment. Soliciting of information allows different products to implement different mechanisms of keeping local cache, for example:

- Multicasting Solicit information messages "on demand" and supporting a full service data base.
- Keeping a cumulative data base on a per-request basis and restricting the number of entries in a data base.
- Processing Response information messages without caching data.
- Using a directed Solicit information message when the provider of services is known or when multicasting is restricted or not permitted at all.

B.1.3 Cluster Static Load Balancing

Clusters of machines might choose to present the same SERVICE_NAME in their multiple multicast messages if they offer equivalent services. By cooperating among themselves to establish a common SERVICE_NAME with appropriate independent SERVICE_RATINGs, cluster members can arrange to share the terminal user load. Digital Equipment VaxCluster present this type of name space to LAT terminal servers.

B.1.4 Multiprocessors, Gateways, Virtual Machines

Multiprocessors may wish to present individual host system processors as unique systems through a shared Ethernet port. More specifically, they may require that messages arriving at the single Ethernet port contain slots all destined for the same physical (virtual) processor.

This can be accomplished by assigning multiple NODE_NAMEs to a single multiprocessor system which shares a single Ethernet port. This will cause a terminal server to establish a new virtual circuit to each different NODE_NAME. Name of a destination node is included in all LAT 5.1 messages to allow addressing of each of the node hidden behind the common Ethernet address. In general destination/source node names and destination/source node addresses allow complete identification of subjects and objects.

Each NODE_NAME can still specify one or more SERVICE_NAMES. This would allow piggybacking of sessions as usual. The same SERVICE_NAME can be assigned to more than one NODE_NAME to achieve static load balancing.

B.2 Compatibility Issues

Compatibility issues between LAT 5.0 and LAT 5.1 based products are discussed below.

B.2.1 Virtual Circuits Establishment

In the LAT 5.0 architecture master is always a subject and slave is always an object of a connection. LAT V5.1 architecture allows both a slave and a master node to be a subject and an object of a connection. A LAT 5.1 master/slave node can use a virtual circuit established in a "wrong" direction to solicit a session over the same virtual circuit, as opposed to starting a new virtual circuit. To achieve this, the NODE_NAME and SYSTEM_NAME presented in the Start message are defined by the LAT 5.1 architecture as follows:

- The NODE_NAME field in the Start message is redefined as SLV_NODE_NAME (name of the slave node).
- the SYSTEM_NAME field in the Start message is redefined as MST_NODE_NAME (name of the master node).

In order to provide compatibility between LAT 5.0 and LAT 5.1 products, implementations of the LAT V5.1 architecture must provide valid slave and master names. The MST_NODE_NAME field received from a LAT V5.0 node must be ignored.

B.2.2 Data_b Slot Length Compatibility

The parameter code field may not be present in some implementations of the 5.0 version. Therefore the presence of this field must be determined based on the slot length.

Some implementations do not include any trailing zero-valued fields. For example, if flow control is being disabled, TOPS does not include the four uninterpreted flow control characters in the data_b slot. However, some implementations do not use the slot length in interpreting slots, which may result in unintended results because the following slot header will be interpreted as slot data. We do not believe any serious incompatibility exists at this time.

If implementations do not process slots using the slot length field, future incompatibility problems may be more severe. For this reason the architecture requires all architecturally specified field to be present in the slot. This simplifies slot processing because all fields normally will be present.

B.2.3 Data_b Slot Data Compatibility

Existing versions of the LAT host implementation use the V5.0 data_b slot format. Therefore, they provide bell-on-discard by enabling HOSTSYNC with null/bell in the START/STOP characters; and provide transparency mode by disabling both HOSTSYNC and TTSYNC. For compatibility with existing implementations, the following rules apply to the setting of flow control, user preference, and transparency mode in a V5.1 node communicating with a V5.0 node.

- LAT V5.1 STOP/START and INPUT/OUTPUT characters can be only XON/XOF. A V5.1 node that receives a data_b slot with the input and output flow control characters specified does not change flow control characters.
- Bell-on-discard:
  - LAT V5.1 implementations must not use null/bell in the START/STOP character.
  - If ENABLE HOSTSYNC is received from a host with null/bell in the START/STOP character, the server sets the port to bell-on-discard and enables HOSTSYNC. In this case, no user_preference parameter field should appear in the data_b slot.
  - If The user_preference parameter field is present, the server should set the user preference to whatever is specified, independent of the HOSTSYNC setting.

- Transparency mode:
  - If DISABLE HOSTSYNC and DISABLE TTSYNC settings came from the host, the server sets the session into passall mode.
  - If passall mode is found in the transparency parameter, it overrides HOSTSYNC/TTSYNC setting.
  - A node implementing V5.1 of the LAT Architecture may set the enable/disable input/output flow control bits (0-3 of control flag) in the 'Set' DATA_B slot to simulate PASSALL and NORMAL mode as shown below:

```
input flow      output flow     desired mode
----------      -----------     ------------
  disable         disable       passall
  enable          enable        normal
```

Table B-2 represents the setting of the port on the 5.1 node by an incoming data_b slot from the 5.0 node (no user preference or transparency mode parameters are present).

Table B–2: Port Setting by Data_b Slots

HOSTSYNC	TTSYNC	Port Setting
Enable (XON/XOF)	Enable	HOSTSYNC/TTSYNC
Enable (XON/XOF)	Disable	HOSTSYNC/NOTTSYNC
Enable (null/bell)	Enable	HOSTSYNC/TTSYNC Bell-on-discard
Enable (null/bell)	Disable	HOSTSYNC/NOTTSYNC Bell-on-discard
Disable	Enable	NOHOSTSYNC/TTSYNC
Disable	Disable	NOHOSTSYNC/NOTTSYNC

B.2.4 Non-Unique Node Names

Server nodes implementing version 5.0 of the architecture do not guarantee uniqueness of the node names (because no master node name concept exists in the 5.0 version of the LAT protocol). To preserve compatibility between 5.1 and 5.0 products, 5.1 node must provide for the case where the same node name with different Ethernet address is used by multiple 5.0 nodes. When this happens, the 5.1 node must create and enter in its data base a unique name for these 5.0 nodes using the Ethernet addresses according to the rules, presented in the section of the document entitled "Specification of names" in the paragraph which describes

creation of the default node names. In other words, the 5.1 nodes must operate compatibly with the 5.0 nodes.

B.2.5 Implementation Of The ethernet And 802 Protocols

The LAT architecture supports both Ethernet and 802 protocols. LAT products can support Ethernet only, 802 only, or both. Some important assumptions made by the LAT architetcure that allow LAT products to use both protocols are:

- Ethernet-only products can discard an 802 message only in software (i.e. may fail to process an immediately following Ethernet message). 802 only products discard an Ethernet message in the hardware. Therefore when messages are transmitted in both formats, the Ethernet message must be sent first.
- Cost of processing either message is the same i.e. if nodes can communicate in both protocols it is not important which one is used.
- Knowledge about datagram's protocol is available to the virtual circuit layer when it operates in both protocols.

Based on these assumptions, the following rules of operation are defined:

- When a node advertises services using the Advertising message it transmits this message in all protocols it supports. If the node can operate in both protocols, it transmits two Advertising messages back to back, the first message in Ethernet format and the second in 802 format.
- When the subject node operates in both protocols and has no information about the protocols supported by the object node, the subject always sends Solicit information, Start, and Command messages in both protocols back to back, the first message in Ethernet format, the second message in 802 format.
- The node that receives one of these messages responds with a message in the same protocol. Note, that if both nodes operate in both protocols, then either one of the protocols can be chosen by the nodes for communication.

C

Algorithm For Assignment/Deassignment Request/Entry Identifiers

Following algorithm is taken from the NSP Functional Specification document (DECnet Digital Network Architecture).

An identifier is a 16-bit value. When a request is queued, an identifier is assigned. When connection actually starts, the identifier is deassined. The algorithm that assigns and deassigns these identifiers is implementation-dependent. There are two requirements for this algorithm:

- It must not assign a given identifier to two entries in the queue concurrently;
- It must not reassign a given identifier 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 LAT 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 identifiers.

C.1 Interface to the Algorithm

The sample algorithm is implemented by a module that accepts three calls: one to assign an identifier, one to deassign an identifier and one to initialize the module.

The following routine assigns an identifier

```
GET-ADDRESS
        returns:  success - an identifier is returned
                  failure - too many identifiers are currently assigned
```

The following routine deassigns an identifier.

```
RELEASE-ADDRESS (address)
        address:  the identifier to be deassigned
        returns:  success
                  failure - identifier was not assigned
```

The following routine initializes the algorithm module.

```
INITIALIZE-ADDRESS
```

The routine is called during initialization and allows the algorithm module to meet the second requirement

C.2 Data Structures

This algorithm forms identifiers of the following form:

```
              random  part     index  part
                 r bits           i bits
where:
     r+i = 16
```

No two concurrently assigned identifiers will contain the same value in the low i bits.

Furthermore, the algorithm restricts the number of identifiers that can be assigned concurrently

```
     2^i-1
```

The data base consists of two vectors and three variables. These are the following.

- Boolean vector INUSE

This vector contains 2^i bits. There is one bit for each possible value in the index part of an identifier. A bit is set to "true" if the corresponding index is in use (i.e., is in the lower i bits of an assigned identifier). The bit is set to "false" otherwise.

- Vector RANDOM

  This vector contains 2^i entries, each r bits wide. An element of the vector contains the random part of the last identifier assigned with the index part equal to the index of this element in the vector.

- Variable NUMBER-ASSIGNED

  This variable contains the number of identifiers 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 identifiers may be assigned.

- Variable INDEX

  This variable contains the index value portion of the last identifier that was assigned.

- Variable TEMP

  This variable is used to temporarily hold the index value portion of an identifier that is being deassigned and in module initialization.

C.3 Algorithm Operation

```
GET-ADDRESS:
   If (NUMBER-ASSIGNED < 2^i-1) then
    Beginwhile
      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 identifier <-- RANDOM(INDEX)
      index part of identifier <-- INDEX
    While (identifier = 0)
      return success
   Else
      return failure
   Endif
RELEASE-ADDRESS:
```

```
   TEMP <-- index part of the identifier
   If (INUSE(TEMP) true
         and RANDOM(TEMP) = random part of identifier) 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
```