Title: Local Area Transport Architecture

File: DELPHI::WRKD$:[MANN.DOC]BEM061.MEM

Date: 21-Dec-82

Author: Bruce Eric Mann

Abstract:

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 tradicional methods of connecting terminals and hosts.

Rev	Description	Author	Date
0.0	Started draft	Mann	20-Dec-82
0.1	Draft state table modifications	Mann,Lauck,Duffy	24-Jan-83
0.2	Finished draft for implementor's review	Mann,Lauck	02-Feb-83
0.3	Updated draft based on Feb 15 review	Mann	03-Mar-83
1.0	Updated draft based on June 3 review	Mann	08-Jun-83
1.1	Updated draft based on June 3 review (DRG first reviewed this version)	Mann	25-Aug-83
1.2	Update based on September DRG review Some new material added.	Mann	19-Dec-83

Change bars have been added since the last review. Not every change has been indicated, but I attempted to mark all the significant changes.

I have suggested an architectural change to the way names are used. Since these changes were not solicited by the DRG, I left the old text in the document flagged with "-" characters, and entered the suggested changed text flagged with the "+" character.

I added a section on implementation issues in service class 1 to describe the use of the names space in different kind of implementations.

The state tables are all marked as changed. I consolidated the two events Out_seq_rcv and In_seq_rcv into a single event Run_rcv, and then changed the action to correspond to this new event. I referenced some interface routines in the state tables to make understanding the action descriptions easier. I added a Stopping state to emphasis the importance of this delay as requested (although I cannot convince myself it is really necessary). I do not believe any different functioning is specified by the new state diagrams.

I added an index.

Some message formats were changed.

CONTENTS

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.

This document presents LAT 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.

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

This 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. This document does not describe any specific issues involved in building products that utilize LAT as a transport service. Those issues 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.

The first five sections present an overview of the architecture.

2 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.

3 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.

A second property of local area networks is dramatically reduced cabling costs. All of the distributed computing resources are connected to a common coaxial cable.

An Ethernet can have as many as 1000 attachments on a single coaxial cable over 1 mile in length. A potential problem in such installations 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.

It is not the goal of the LAT architecture 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 Ethernet. This eliminates the need for any routing capability.
- the nature of the communication is inherently asymmetric. This simplifies connection management 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.

These 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.

It is not a goal of this architecture 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.

4 TERMINOLOGY, ASSUMPTIONS AND NOTATION

- 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.

- host (operating system) - a special case of an integrated set of server systems which implement an operating system. The use of the term "host" in this context is refering to the operating system an not to the host services.

- host (application services) - Named services offered by applications. There can be many such application services associated with each host operating system.

- port - the adapter between a processor and interconnect which implements the Ethernet data link layer.

- 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.

- virtual circuit - a logical stream of data, established between a terminal server and a host with the characteristic that data delivery is bidirectional, sequential, timely, and error-free. On Ethernet, a virtual circuit service is a value added service since the Ethernet data link provides a datagram service.

- message - a datagram under virtual circuit error control.

- slot - a segment of a message used to commuicate data between a terminal on a terminal server and a host application. 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 utilizing an underlying 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.

LAT assumes the Ethernet has very predictable attributes. LAT's performance depends on "low probability" events occuring 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)

All numeric values are specified in decimal.

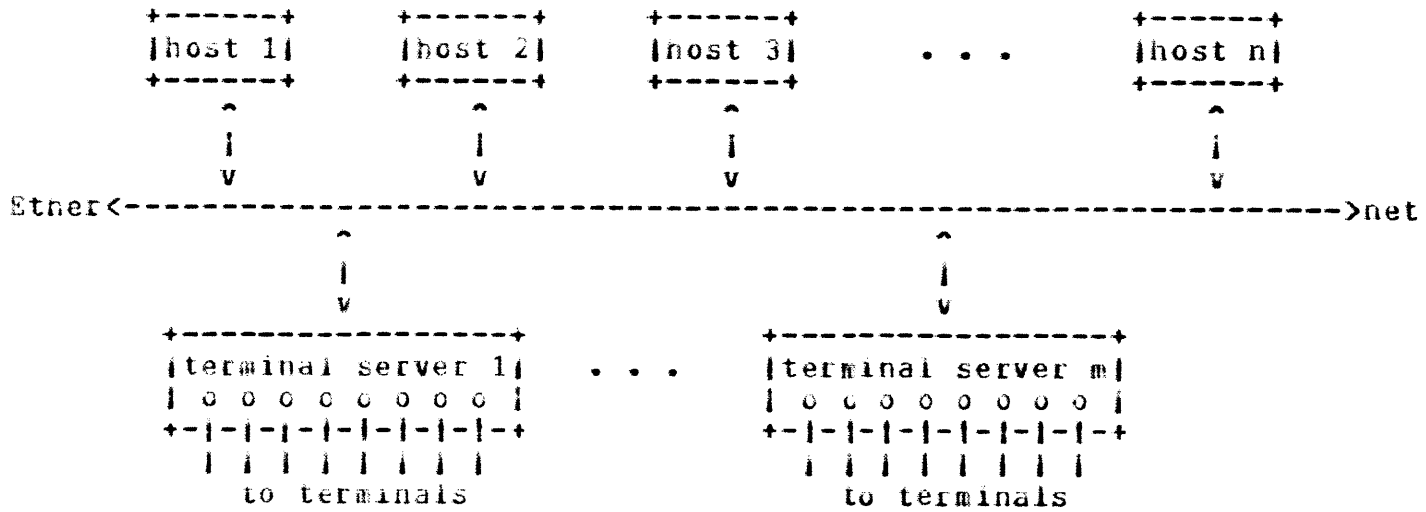
Character string literals are quoted as in "DELPHI". Occasionally, phrases and terms that are conceptually important to the architecture are quoted, "balanced mode" for instance.

Capitalized names are architecturally defined, an example is SERVER_CIRCUIT_TIMER. Lower case names are function names or events. For example: transmit_unacknowgledged_queue is a function name and Send_data is an event. Many of these names appear in the document index.

5 OVERVIEW OF ARCHITECTURE

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

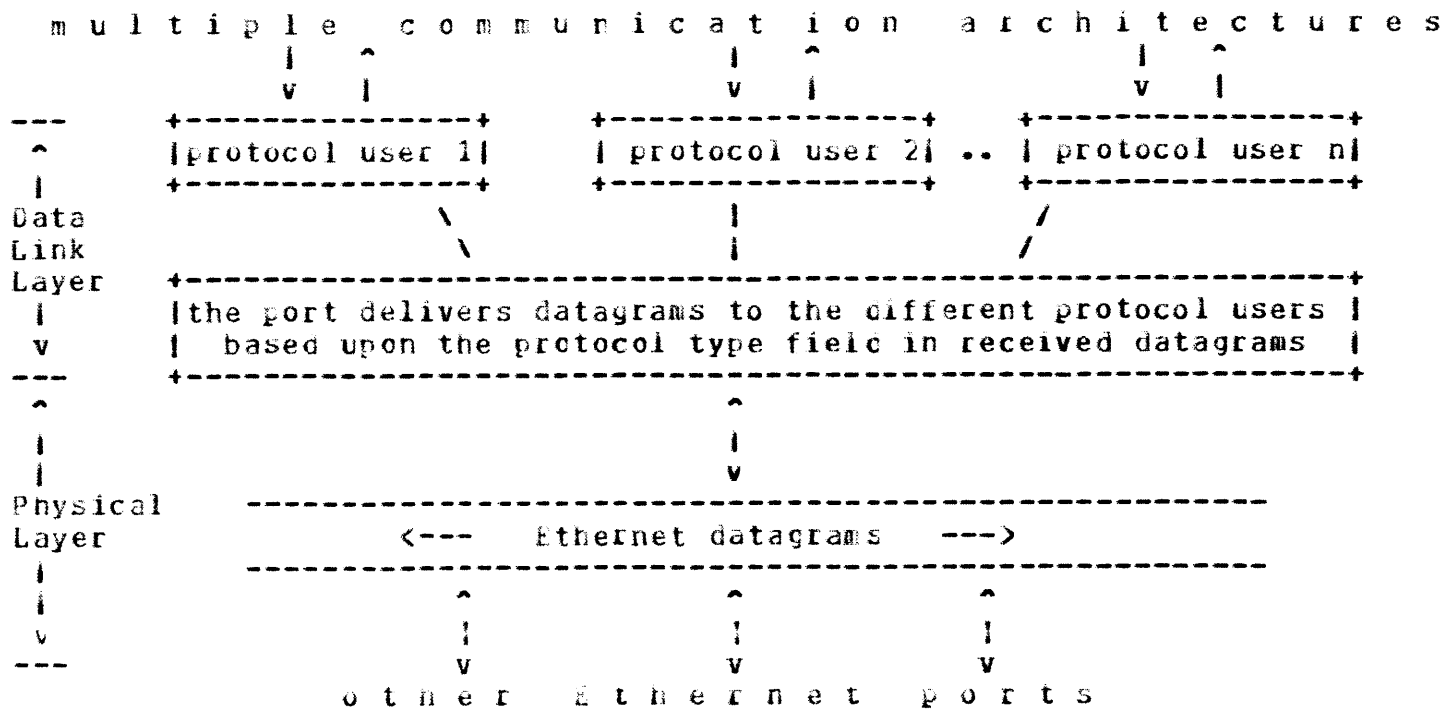
The principal capability provided by the architecture is the logical connection of terminals to host processors. A vertical view of the topology would reveal:



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

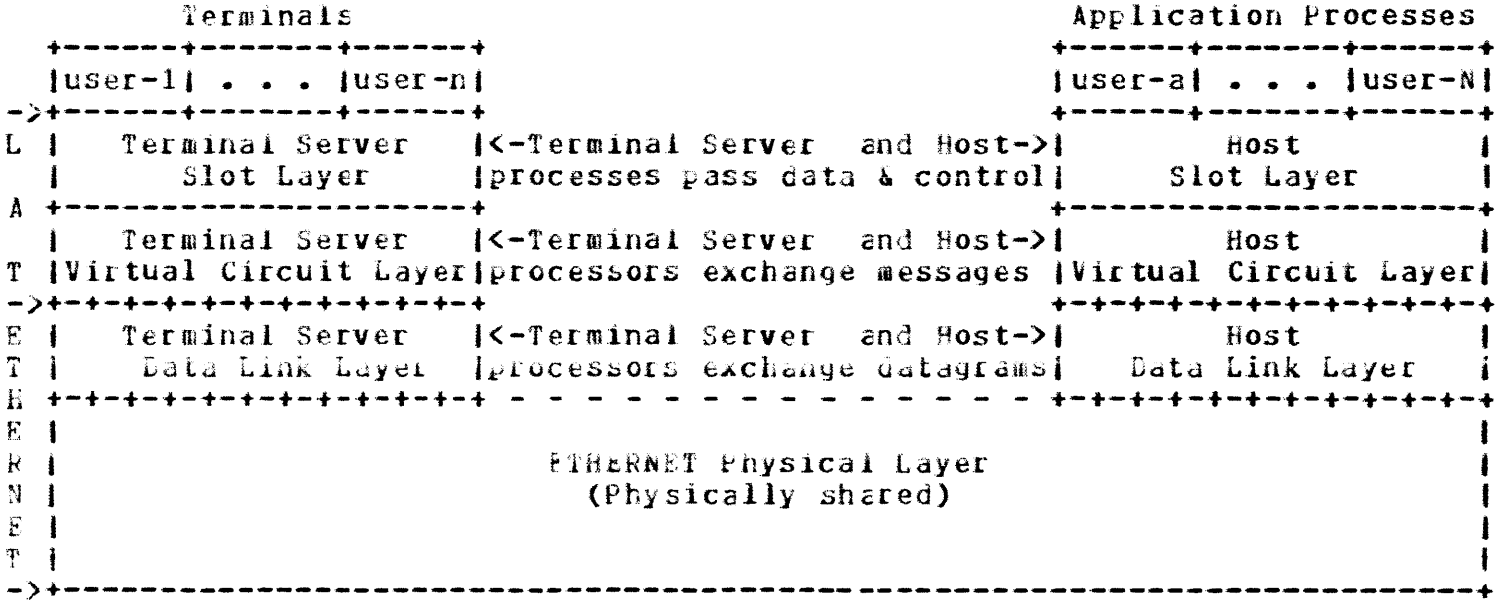
It is possible for a single system to operate simultaneously as both a host and a terminal server. If such a system wishes to allow local terminal users to connect to the local host system, Ethernet messages transmitted by the local system to itself must be simulated by the local Ethernet port driver since these messages will not be delivered by the Ethernet data link.

The Ethernet itself is layered:



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 implementaion of the LAT architecture. The LAT architecture could be viewed as a layered architecture:



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

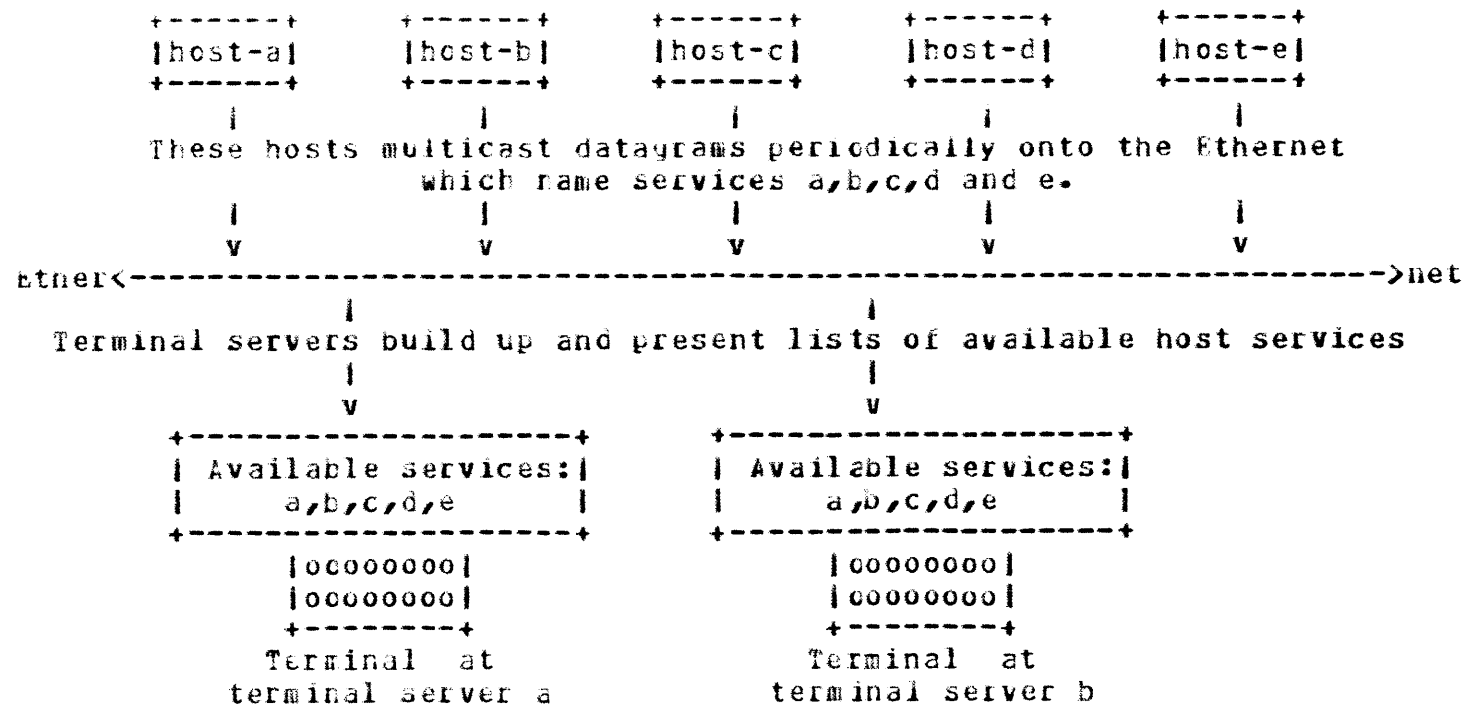
Functionally, the virtual circuit layer establishes and maintains a shared virtual circuit. The slot layer multiplexes one or more users connections over the underlying shared virtual circuit.

5.1 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 applications.

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.

In a terminal server application (see appendix a), each host makes the services it provides known to the local area by advertising the service in a datagram periodically multicast to all terminal servers. These services are represented as names in each multicast datagram (SLOT_NAMES). The terminal servers receive these datagrams, build up a list of available hosts and host services and offer the user a selection of these hosts services. Normally, these services would be presented at each terminal in a console mode local to the terminal server as shown in the diagram:



A user then simply selects the desired host service from the displayed list of available host services. The slot layer translates the selected (SLOT_NAME) into the name of a host which contains the service (CIRCUIT_NAME). After successfully establishing a session with the desired host service, further input typed by the user at the terminal is transferred to the host, just as though the terminal were, in fact, local to the host. Thus the peculiarities (such as meaning of control-C, control-T, control-Y ...) of a particular host system or host application process remain unknown

to the terminal server.

For network management purposes, each terminal server can present a different set of available host services based on group codes assigned to hosts 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 on "LOCAL AREA DIRECTORY SERVICE" for more details.

5.2 Virtual Circuit Layer

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

The virtual circuit allows messages to be reliably exchanged between the terminal server and the host. The format and the rules governing the exchange of these messages is specified by the Local Area Transport
+ architecture. The virtual circuit layer is responsible for translating 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 is timer based. The host always responds to these messages from the terminal server. Under certain conditions (see state diagram) the host may send an unsolicited message.

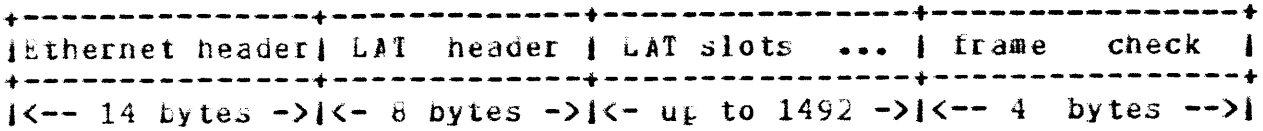
The architecture minimizes the overhead of the virtual circuit by:

- using simple, asymmetric connection management - Only one virtual circuit is established between any pairing of a terminal server and host. The virtual circuit service is always initiated at the request of the terminal server.

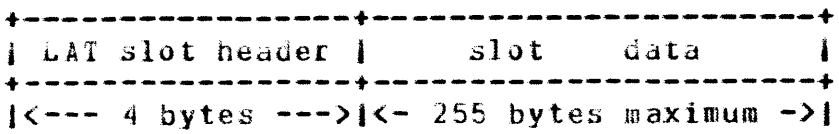
- procrastinating so there is more to do when things have to be done - messages are not normally exchanged when data is available to be transmitted. Instead, messages are exchanged periodically. The rate of exchange can be set to a constant value or varied to suit the needs of the application. A typical 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.

  An Ethernet message is limited in length to 1518 bytes:



  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 and, because messages can be exchanged quickly, does not reduce the available bandwidth below useful levels for most applications.

5.3 Product Considerations

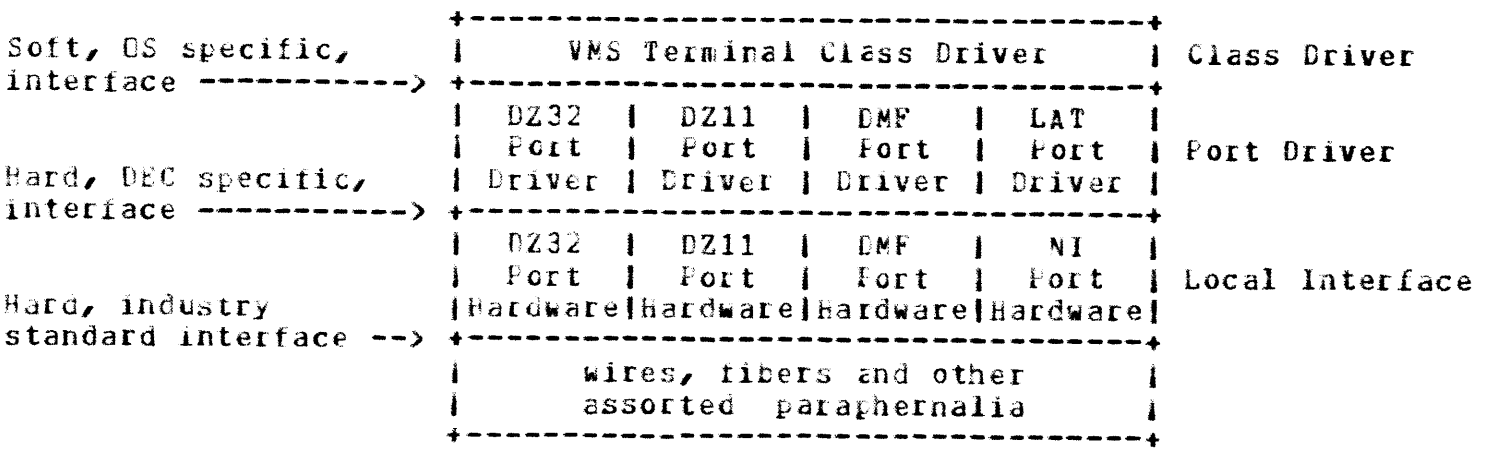
5.3.1 Host -

Hosts implement the passive (or slave) end of the virtual circuit because this end of the virtual circuit is simpler (and therfore 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 follows:

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

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 VAX/VMS operating system:



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.

5.3.2 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 connect to a host system.

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.

6 LOCAL AREA DIRECTORY SERVICE

The directory service is based on the multicast mechanism built into the Ethernet. The directory service is designed to minimize the need for intervention by 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 is available within milliseconds of the host's announcing the service. If a host should crash, the terminal servers can notify the users within a few seconds that the host may have crashed.

The overhead associated with the maintenance of a directory service is small compared with the overhead of the connection management and data transport services.

Host and terminal server addressing (i.e., the directory service) is independent of the rest of the architecture. In principle, the function is not necessary to the proper operation of the architecture. If the terminal servers had built-in knowledge of host Ethernet addresses and the classes of service offered by each host, the directory service would be redundant.

The virtual circuit layer must translate names into Ethernet 48-bit addresses. This translation is normally accomplished by utilizing data received by the terminal server in multicast messages.

While the different classes of service share the underlying data transport service, each class of service defines a different directory service appropriate to the needs of that particular class of service. This architecture is described in the context of terminal servers and hosts. A directory service that partially fulfills the needs of the hosts and terminal servers is described in detail in the appendix titled "SERVICE CLASS 1 - INTERACTIVE TERMINALS".

6.1 Two Level Name Space

Names supplied by multicast messages are one of two differnt types:

- CIRCUIT_NAME - Circuit names correspond to "service access points" (SAPs) or "sockets" in the host virtual circuit layer. Each virtual circuit between a host and a Terminal Server connects two of these virtual circuit layer service access points together. Since the Terminal Server originates all virtual circuit connections, the host SAPs are named and the Terminal Server SAP are not named. The virtual circuit assumes the name of the host SAP. A host can have two different circuit blocks with the same CIRCUIT_NAME - as long as these two different circuit blocks are associated with a different Terminal Server (a different REM_ADDRESS value in each circuit block).

+ o SLOT_NAME - Slot names correspond to service access points in the slot layer in both the terminal server and in the host. These names are supplied by and to users for the purpose of providing a convenient means of identification. These names are especially useful in establishing slot sessions when more than one type of service is offered by a host system.

Multicast messages are not required to specify a CIRCUIT_NAME. These messages are treated identically to messages that do specify CIRCUIT_NAMES (as shown below).

Systems that wish to announce an available service send multicast messages periodically.

Multicast messages containing these SOURCE_ADDRESSES (SA), CIRCUIT_NAMES (CN), SLOT_NAMES (SN) names and SLOT_RATINGS (numbers):

ABCD-SA	BCDE-SA	CDEF-SA	DEFA-SA
DELPHI-SN-11 VMSDEV-SN-24 VMSMAIL-SN-2	STAR-CN STAR-SN-234 METEOR-CN METEOR-SN-255 VMSDEV-SN-255	GALAXY-CN	
DELPHI is a part of VaxCluster A	METEOR is a part of VaxCluster A	GALAXY is a part of VaxCluster B	ATHENS is a part of VaxCluster C

Would cause the following database to be constructed in a sever:

CIRCUIT_NAME	SLOT_NAME	SLOT_RATING	Host source addr
			ABCD
	DELPHI	11	
	VMSDEV	24	
	VMSMAIL	2	
STAR			BCDE
	STAR	11	
METEOR			BCDE
	METEOR	255	
	VMSDEV	255	
GALAXY			CDEF
			DEFA

And the following display to a server user:

```
   DELPHI     DELPHI is a part of VaxCluster A
   VMSMAIL    DELPHI is a part of VaxCluster A
   STAR       METEOR is a part of VaxCluster A
   METEOR     METEOR is a part of VaxCluster A
   VMSDEV     DELPHI is a part of VaxCluster A
(  or         VMSDEV     METEOR is a part of VaxCluster A  but not both )
```

Notice that the CIRCUIT_NAMEs "DELPHI" and "GALAXY" are not displayed to the user. The services on node "GALAXY" and the system sending the multicast message from Ethernet port "DEFA" are unusable since they are not displayed.

6.2 Service Class 1 Multicast Message

Service class 1 utilizes a single multicast message. This message enables the virtual circuit layer to translate CIRCUIT_NAMEs into 48-bit Ethernet destination addresses and the slot layer to translate SLOT_NAMEs into CIRCUIT_NAMEs. The format and useage of this Service Class 1 message is described in the Service Class 1 appendix.

7 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.

7.1 Slot Data

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

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

7.2 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.

7.3 Virtual Circuit Service

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

7.3.1 Virtual Circuit State -

The state of a virtual circuit is captured in a data structure called the Circuit Block. A separate Circuit Block is maintained by the host and by the terminal server. 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.

+
+ o CIRCUIT_NAME - the name of the virtual circuit

o REM_ADDRESS - the Ethernet address of the remote system.

o LOC_ADDRESS - the Ethernet address of the local system.

o 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.

o 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.

o REM_CIR_ID - Remote circuit identification

o LOC_CIR_ID - Local circuit identification (index to circuit block itself)

| o 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...).

| o 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.

o 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. 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_CIRCUIT_TIMER (host only) - The host's circuit timer is an interval timer which is started when the host 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 expires approximately every 80-100 milliseconds. This timer is used to initiate the transmission of new data but is not used to retransmit unacknowledged data. The terminal server retransmit policy is explained in the AXIOMS and ALGORITHMS section in the Message Trasmitter section.

- HOST_RETRANSMIT_COUNTER, SERVER_RETRANSMIT_COUNTER - Count of number of times the current message number has been retransmitted. If this value reaches 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 of the virtual circuit state through Halted; even users that attempt to disconnect would be "hung" in the disconecting sub-state until a message was sucessfully 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. Quality values are implementation dependent.

- Quality should be be considered excellent if the ratio of the (total of retransmited and duplicate messages)/(total messages transmitted and received) remains under 1/10,000 for a particular virtual circuit.
- Quality should be considered good if this ratio remains under 1/1,000.
- Quality should be considered poor if this ratio exceeds 1/1,000.
- Quality should be considered unacceptable if the value of the retransmit counter (either HOST_RETRANSMIT_COUNTER or SERVER_RETRANSMIT_COUNTER) reaches LAT_MESSAGE_RETRANSMIT_LIMIT.

- 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.)

7.3.2 Architecturally Controlled Names And Variables -

7.3.2.1 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.

- NXM1 - 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.

7.3.2.2 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.

7.3.2.3 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 courters.

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.

Implemetations must collect and be capable of displaying the following list of values for each system:

- Those specified in "Digital Ethernet Node Product architecture"?
- Total number of illegal messages received (and associated remote system if possible)
- Total number of illegal slots received (and associated remote system if possible)

NOTE

Illegal message and slot counters are maintained as a part of the virtual circuit database listed below. If these counter databases are retained for some time after the virtual circuits are terminated, a valuable piece information is retained to diagnose the source of the illegal data - the name of the remote system !

Implemetations must collect and be capable of 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
- DUPLICATES_RECEIVED
- ILLEGAL_MESSAGES_RECEIVED (causing the virtual circuit to be aborted)
- ILLEGAL_SLOTS_RECEIVED (causing the virtual circuit to be aborted)

Base of use and maintenance would indicate that the following counters should also be displayable by each system (may be required for a Digital implementation):

- TOTAL_MESSAGES_TRANSMITTED (excluding multicast)
- TOTAL_MULTICAST_DATAGRAMS_TRANSMITTED
- TOTAL_MESSAGES_RECEIVED (excluding multicast)
- TOTAL_MULTICAST_DATAGRAMS received
- total of other datagrams received (some implementations may not need this counter if the Ethernet data link implements a perfect filter).
- TOTAL_FRAME_CHECK_ERRORS received
- TOTAL_FRAMES_DEFERRED due to busy Ethernet
- TOTAL_LENGTH_ERRORS (discarded)
- TOTAL_DUPLICATE_CIRCUIT_NAMES received

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

7.3.2.4 Error Handling - Illegal Slots And Messages -

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 session. These messages and slots should not occur, but if they do:

- An error must be logged. 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.
- The message or slot should be discarded.
- The (underlying) virtual circuit should be terminated.

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:

- A received message with either the DESTINATION_ADDRESS or the SOURCE_ADDRESS equal to zero.
- An unknown MSG_TYPE in a received message.
- A non-zero SRC_CIR_ID in a received Stop message.
- A zero SRC_CIR_ID in a received Start message.
- A non-zero DST_CIR_ID in a Start message received by a host.
- A zero DST_CIR_ID in a Start message received by the terminal server.
- A zero DST_CIR_ID in a Run message.
- A zero SRC_CIR_ID in a Run message.
- Others - please get them added to this list.

Examples of illegal slots are:

- An unknown SLOT_TYPE value is received.
- An unknown SERVICE_CLASS is received in a Start slot.
- A non-zero SRC_SLOT_ID in a received Stop slot.
- A zero SRC_SLOT_ID in a Start slot.
- A non-zero DST_SLOT_ID in a Start slot received by a host.
- A zero DST_SLOT_ID in a Start slot received by a terminal server.
- An Attention slot specifies a non-zero value in the credit field (documented as an MBZ field in the message format section).
- a Start slot received in the Run state (without and intervening Stop slot)
- a Reject slot received in the Run state.
- 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).

- a Run slot with a zero SRC_SLOT_ID.
- a Run slot with a zero DST_SLOT_ID.
- Others - please get them added to this list.

7.3.2.5 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 - AB-00-03-00-00-00-00 (hexadecimal) or as a bit stream, first bit on Ethernet at left (1101.0101.0000.0000.1100.0000.0000.0000.0000.0000.0000.0000)

The following values must be specified before an implementation is operational. If the values are settable within the ranges specified, the names used to refer to the parameters must be reasonable facsimile of the name used below. The architecture requires the values be within the ranges specified:

- PROTOCOL_VERSION - The value 1.
- CIRCUIT_NAME - must be specified by the system manager before the start of service can be announced.
- SERVER_CIRCUIT_TIMER - In the range 10-900 milliseconds (80 milliseconds recommended).
- HOST_CIRCUIT_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 (HOST_RETRANSMIT_COUNTER limit).
- HOST_MULTICAST_TIMER - In the range 10 to 180 seconds (30 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 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.
- LAT_MAX_SERVERS - The number of data link buffers dedicated to starting new virtual circuits. A value of 1 is recommended.
- NBR_DL_BUFS - The number of data link buffers assigned minus one. A value of zero is recommended.
- PRODUCT_TYPE_CODE - Start messages containing the value zero in the PRODUCT_TYPE_CODE field may be rejected by an implementation. The REASON given will be that the product type code is unsupported or unknown.

  1. FLOIF terminal server

2. POSEIDON terminal server
3. VAX/VMS host
4. RSX11-M host
5. RSX11-M+ host
6. TOPS-20 host
7. TOPS-10 host
8. TOPS-20 terminal server
9. LAT-11 terminal server
10. Berkley/UNIX host

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

+ o SLOT_NAME - the name of the host service displayed to the user.
o LAT_KEEP_ALIVE_TIMER - In the range 10 to 255. A value of 20 seconds is recommended.

The parameters that are optional are:

o PROTOCOL_ECO - The value 0 unless an ECO has been applied.
o FACILITY_NUMBER - A value supplied via the local command interface.
o SERVER_NAME - A name supplied via the local command interface.
o LOCATION_TEXT - Text supplied via the local command interface.
o 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.

7.3.3 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 or server crashes.

7.3.4 Virtual 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. Timer - the virtual circuit timer expires.

10. Resend_limit - The retransmit counter reached the limit LAT_MESSAGE_RETRANSMIT_LIMIT.

11. 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.

7.3.5 Response Requested Flag And Balanced Mode -

When the most recent message received from the host 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 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 unecessary 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 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_CIRCUIT_TIMER expires. This sub-state is exited when the host receives a message that acknowledges all of the currently unacknowledged messages. At this point and the HOST_CIRCUIT_TIMER is stopped. The purpose of this sub-state is to guarantee "unsolicited" message delivery.

7.3.6 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.

- The LAT virtual circuit layer maps messages onto circuit blocks based on the value of the SOURCE_ADDRESS (48 bits) field of the received message. A match to a circuit block is found if the SOURCE_ADDRESS field of the received message equals the REM_ADDRESS field in the circuit block.

+ The LAT virtual circuit layer maps Start messages onto circuit blocks based on the value of the CIRCUIT_NAME field and the SOURCE_ADDRESS field of the received Start message. A match to a circuit block is found if the CIRCUIT_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. If no match is found, a new circuit block can be created and these two values used to initialize the new circuit block. If no CIRCUIT_NAME is supplied in a Start message (the CIRCUIT_NAME_LENGTH is zero), then the SOURCE_ADDRESS of the message alone is used to map the message onto a circuit block.

- However as an optimization of CPU utilization, Run messages may be 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. (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.)

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

modulo 256).

- o Inv_run_rcv - DST_CIR_ID of message not equal to LOC_CIR_ID in circuit block or SRC_CIR_ID of message not equal to REM_CIR_ID of circuit block or either field in the message is equal to zero.
- o Stop_rcv - DST_CIR_ID of message equals LOC_CIR_ID of circuit block and SRC_CIR_ID of message equals zero.
- o Inv_stop_rcv - DST_CIR_ID of message not equal to LOC_CIR_ID in circuit block or SRC_CIR_ID of message not equal to zero.

In the case of the first Start_rcv event in the host, 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.

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

7.3.6.1 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.	Running
	Resend_limit	Notify users, send Stop.	Halted
	Timer	Resend Start.	Starting
	Stop_rcv	Notify users.	Halted
	VC_halt	Send Stop (with reason).	Halted
	Inv_stop_rcv	No actior.	Starting
	any other msg	Process Start, send Start.	Starting

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.

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_XMIT<- <empty>
13. RRF flag is cleared
14. DWF is cleared

terminal server virtual circuit state table - continued

State	Event(s)	Action(s)	Next State
Running	Run_rcv	If msg is out of sequence: zero NBR_SLOTS in msg hdr. If RRT flag is set: set DWF. Process received ack; process message.	Running
	Timer	If messages remain unacknowledged: resend all uracked messages. 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 (with reason)	Halted
	Stop_rcv	Notify users	Halted
	any other	Set DWF	Running

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

7.3.6.2 Host Virtual Circuit State Table -

State	Event(s)	Action(s)	Next State
Halted	Start_rcv or Inv_start_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
	Resend_limit	notify users (or optionally halt via VC_halt event).	Starting (Halted)
	VC_halt	Send Stop (with reason)	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

"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 CIRCUIT_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. CIRCUIT_NAME<- <copied from the receive Start message>
2. REM_ADDRESS<- <SOURCE_ADDRESS from the received Start message>
3. LOC_ADDRESS<- <value assigned to the local system>
4. REM_CIR_CID<- <copied from received Start message>
5. LOC_CIR_ID<- <unique connection id>
6. NXMT<- 0 - Next message number to transmit
7. ACK <- 0 - most recent message number received in sequence
8. LXMT<- 0 - Lowest unacknowledged message number transmitted
9. HXMT<- 0 - Highest unacknowledged message number transmitted
10. HOST_CIRCUIT_TIMER<- <stopped>
11. HOST_RETRANSMIT_COUNTER<- 0 (counts up to LAT_MESSAGE_REXMIT_LIMIT)

12. UNACKED_XMTQ<- <empty>
13. RRF flag is cleared
14. DWF is cleared

host virtual circuit state table - continued

State	Event(s)	Action(s)	Next State
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
	Send_Data	Start timer; queue_transmit_message; transmit_unacknowledged_queue.	Running
	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

"Start 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 timer expires, most likely the terminal server has crashed or the Ethernet data link has failed.

The queue_transmit_message function conditionally modifies the RRF in the Circuit block before the message header is generated. The RRF flag is always set in the Circuit Block if the Circuit Block DWF is set or if the last transmit message buffer is being consumed. The RRF flag is cleared whenever the queue_transmit_message function finds that the Circuit Block DWF is clear and the last transmit message buffer is not being consumed.

7.4 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.

7.4.1 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.

7.4.2 Host Session Management -

The host implementation may choose to always accept or reject a connection (respond to a start slot received by the host with a start or reject slot) based soley 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 application to make a decision to reject the connection. Interactive terminal login is an example of such a host application. The account name and password must be supplied within the context of the session before the host application can reject the session.

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

7.4.3 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 divicing 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.

7.4.4 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.

7.4.5 Illegal Slots -

If an illegal slot is received, the virtual circuit on which the slot was received should be stopped (the slot session, if it is identifiable, MUST be stopped). An error log entry should be made.

7.4.6 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:

+ o REM_SLOT_NAME - the name of the local slot block
+ o LOC_SLOT_NAME - the name of the local slot block
o REM_SLOT_ID - Remote slot connection identification
o LOC_SLOT_ID - Local slot connection identification
o 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.
o LOCAL_CREDITS - available credits to transmit slots. This field is initialized to zero when the slot block is created. The slot layer slot demultiplxer 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.
o SLOT_TYPE - slot type. One of Start, Data_a, Data_b, Attention, Reject or Stop.
o DRF - Data Ready Flag. Set whenever slot data is available. Cleared by the slot layer whenever all slot data is under virtual circuit control. This flag is not essential to the architecture.
| o SLOT_COUNT - byte count of next field (which could be zero)
| o SLOT_DATA_BUFFER - This buffer is used to store Data_a received over
| the session as the result of extending slot credits.
o 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 message under virtual circuit control.

7.4.7 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	0	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.

7.4.8 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
	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.

7.4.9 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.

7.4.10 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
	Stop_rcv	No action.	Halted
	Reject_rcv	No action.	Halted
	Run_rcv	Process slot.	Running
	User_data or User_rcv	Set DMF and DMP.	Running
	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 user at the host (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.

8 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).

8.1 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 transfered in attention slots.

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

- status and control information (Data_b) can be transfered 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.

- o 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

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

8.2 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.

8.2.1 Summary Of Functions -

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

Function offered:	Type of function:
S- start_service_class	directory service
S- poll_service_class	directory service
S- start_session (connect)	session service
S- new_session_poll	session service
S- accept_new_session	session service
S- 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
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

8.2.2 Description Of Functions -

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

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

- o S-start_session(address,class,min_slot_size;handle,status),
+
+ o S-start_session(slot_name,class,min_slot_size;handle,status),
  SM-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.
  - o address - Ethernet address of host
  + o slot_name - The name of the remote slot block.
  - o class - service class
  - o handle - used to reference the session locally
  - o slot_size - the minimum slot size queued by queue_rcv_slot_buffer for Data_a, Data_b and Attention slots.
  - o status - one of: request_active, request_rejected (see service class appendix), insufficient resources to complete request, no such session.
  - o 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.

- o M-new_session_poll(;handle,status),
  M-accept_new_session(handle,min_slot_size,mode;status),
  M-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
- 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).

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

8.3 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.

8.3.1 Summary Of Functions -

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

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

8.3.2 Description Of Functions -

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

- o S-VC_start(address,max_sessions;handle,status),
- o S-VC_start(circuit_name,max_sessions;handle,status), H-accept_virtual_circuit(handle;status), SH-VC_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 system must not dally in responding to a request to start a virtual circuit. If the host wishes not to have any new circuits established, all service classes should be disabled.
  - o address - Ethernet address of destination system
  - o circuit_name - The name of the virtual circuit.
  - o 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.
  - o handle - handle used to reference the virtual circuit locally (a reference to the circuit block)
  - o status - one of: insufficient resources to complete request, no such circuit.

- o 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.
  - o handle - handle used to reference the virtual circuit locally
  - o status - one of: Starting, Running or Halted (with reason).
  - o quality - the virtual circuit quality, one of: VC_ok, VC_suspect, Ethernet data link disabled.

- o 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.
  - o buffer - address and length of a buffer.
  - o 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

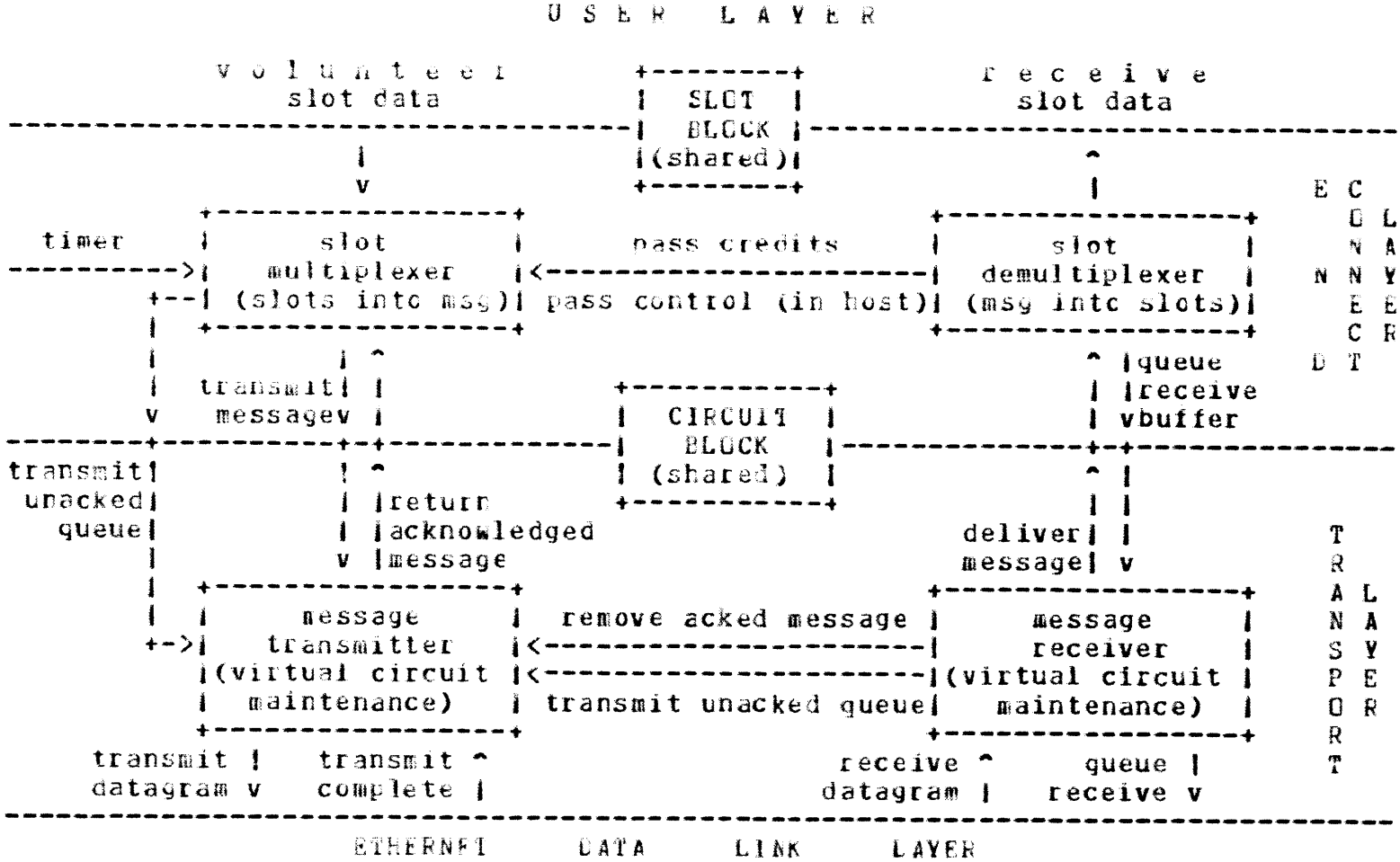
- SH-poll_receive_message_done(handle;buffer,status) - this function allows the slot layer poll the transport layer for any received messages.

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

9 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:



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

9.1 Virtual Circuit Layer

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

9.1.1 Circuit Starter (terminal Server Only) -

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

- o 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.
- o queue the receive message buffer(s) to the Ethernet data link via the queue_rcv_datagram function
- o store the transmit message buffer in the circuit block
- + o Translate the CIRCUIT_NAME into a 48-bit Ethernet address and load
- + this address into the REM_ADDRESS field of the Circuit Block.
- o generate a Start message (using the transmit message buffer) and queue it to the Message Transmitter via the queue_transmit_message function

9.1.2 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 might also send an extra multicast datagram to indicate that service has been ended.

9.1.3 Message Receiver -

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

- o Ethernet destination address of the frame matches that of the local system
- o 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 (host) -
  - 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 no 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 CIRCUIT_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 teminal 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 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, stop the circuit 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 and terminal server are out of synchonization 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 intc 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.

9.1.4 Message Transmitter -

The terminal server message transmitter normally maintains an unacknowledged transmit queue of one entry, while the host 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_RETRANSMIT_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) clearing the RRF flag if the Circuit Block DWF is clear and this is not the last transmit message buffer; setting the RRF flag if the DWF is set or this is the last transmit message buffer.
  - (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 occurs at the rate HOST_CIRCUIT_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 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 approximate one second intervals after the original message was sent. Thereafter, messages should be retransmitted at one second intervals.

NOTE

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

9.1.5 Circuit Timer Policy -

The circuit timers, in both the terminal server and the host, 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.

9.1.6 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 duplicates, 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.

9.2 Slot Layer

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

9.2.1 Administrator -

9.2.1.1 Host -

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 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 circuit timer (no circuit timer could be active since no virtual circuits are active).

9.2.1.2 Terminal Server -

When service is started in the terminal server:

- o at least one buffer should be queued to the data link to receive multicast addresses.
- o start the 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.

9.2.2 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 SLOT_NAME supplied in the start_session function into a CIRCUIT_NAME to be passed in the VC_START furction when a new virtual circuit must be established.

9.2.3 Session Starter (host) -

A host 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 user this choice, but instead accepts or rejects the session without notifying the user. Later the user may stop the session with a reason.

If supplied with a SLOT_NAME, the host session can be bound to the specific application process associated with the SLOT_NAME.

9.2.4 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 SLOT_NAME supplied in the Start slot to bind the
+ session to a particular service access point in the host.

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.

9.2.5 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 reponse. 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 termial 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, Attenton data or REMOTE_CREDITS waiting to be transfered.

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 circuit 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.

9.2.6 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 application process or the terminal server user can issue an end_session (disconnect) function call.

9.2.7 Flow Control -

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

9.2.7.1 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.

9.2.7.2 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 acknowledgement 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.

9.2.8 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 Run and Stop 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.

If a change will make systems incompatible in the field, then a newer protocol version number is allocated.

9.3 Other Processes

9.3.1 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.

9.3.2 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.

10 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.

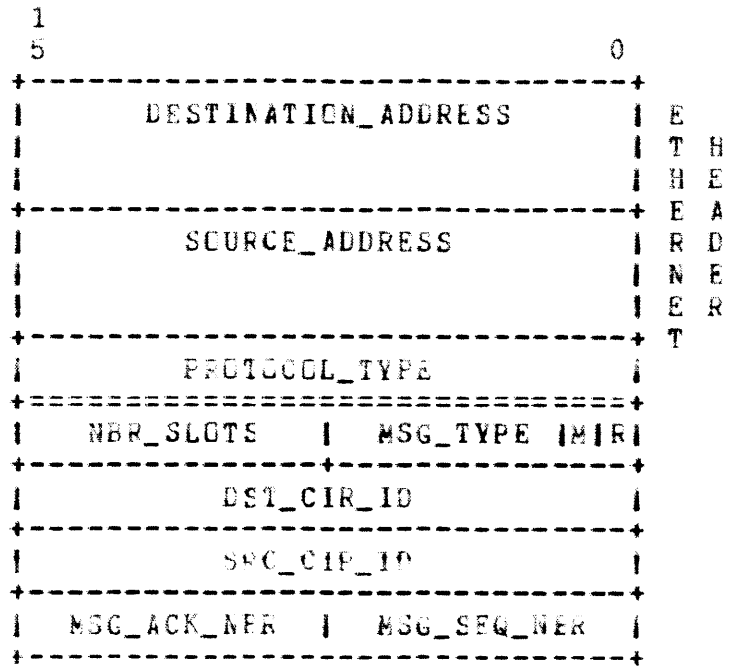
Fields are represented as bit streams, right to left. All fields are an integer multiple of eight bits. The symbol "=" is used to indicate fields of varying or indeterminate length.

A message transmitted from a master to a slave (from a terminal server to a host) 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.

LAT messages must be padded to the Ethernet 64 byte minimum. The data used to pad the frame to achieve this 64-byte minimum are unpredictable.

10.1 Virtual Circuit Message Header

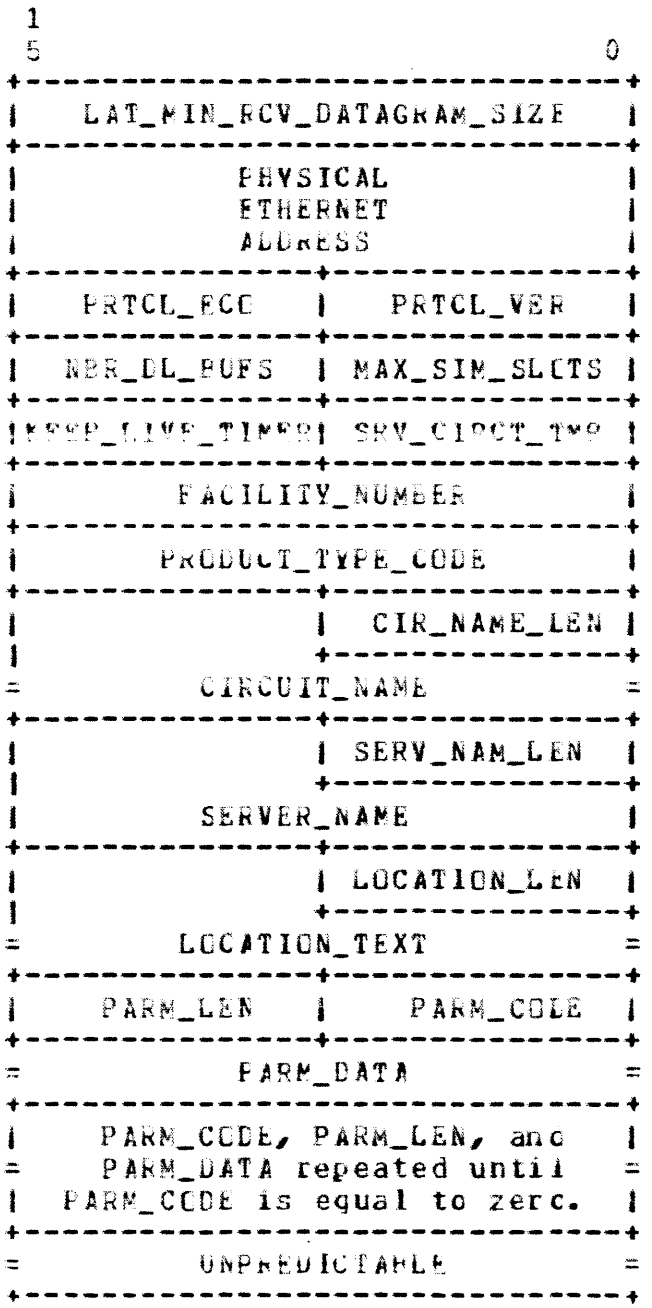
All messages have the same header format:



- R (1 bit) - RRF flag. This flag is clear for all message types except for Run messages transmitted from the host 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. Messages sent by the host 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
- MSG_ACK_NBR - message acknowledgement number

10.1.1 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 must specify these same two fields as non-zero.



- LAT_MIN_RCV_DATAGRAM_SIZE (2 bytes) - The minimum and maximum sizes of frames are restricted by the Ethernet specification to be in the range 46 through 1518 bytes. An implementation must specify the minimum size receive buffer that it queues to the data link layer of Ethernet. Legal values are restricted by the LAT architecture to be in the range of 576 through 1518 bytes.

- PHYSICAL_ETHERNET_ADDRESS (6 bytes) - The unique value assigned to the port hardware.

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

- PRTCL_ECO (1 byte) - The protocol version ECO (Engineering Change Order) of this message and of all messages transmitted during this session. For any given protocol version, ECOs are backward compatible. The PRTCL_ECO level is intended to be used to reflect patches made in the field by automatic updates or by field software specialists.

- 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 milliseconds. Specified by terminal server. This field is ignored when received from the host. Legal values for the circuit timer are in the range 10 to 150 milliseconds.

- KEEP_ALIVE_TIMER (1 byte unsigned) - Value specified in seconds by terminal server. This field is ignored when received from the host by the termial server.

- 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.

- PRODUCT_TYPE_CODE (2 bytes unsigned) - The product type codes are assigned by Digital Equipment Corporation on demand.

+ - CIRCUIT_NAME_LEN (1 byte unsigned) - The byte count of the next field.

+ o CIRCUIT_NAME - The name of the virtual circuit.

o SERV_NAME_LEN (1 byte unsigned) - Byte count of SERVER_NAME field.

o SERVER_NAME (SERV_NAME_LEN bytes) - The text within this field should describe the name of the terminal server or host.

o LOCATION_LEN (1 byte unsigned) - Byte count of LOCATION_TEXT field.

o LOCATION_TEXT (LOCATION_LEN bytes) - The text within this field should describe the physical location of the terminal server or host.

o 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.

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

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

10.1.2 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.

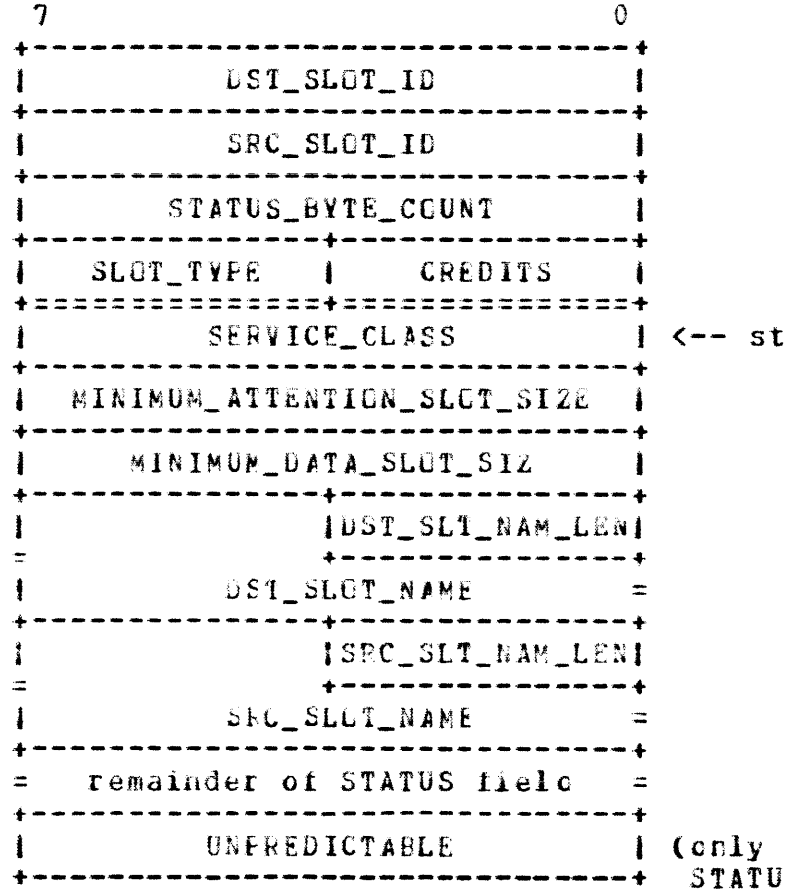
Each slot is aligned on word (16-bit) boundaries. The first slot is contiguous to the message header. The second slot is contiguous to the first if the slot's total length is even. If a slot's total length is odd, then one byte of UNPREDICTABLE data is used as a pad byte between the slot to force the 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 alway zero or positive and credits are coveyed by Data_a slots as a byte value.

10.1.2.1 Start Slot -

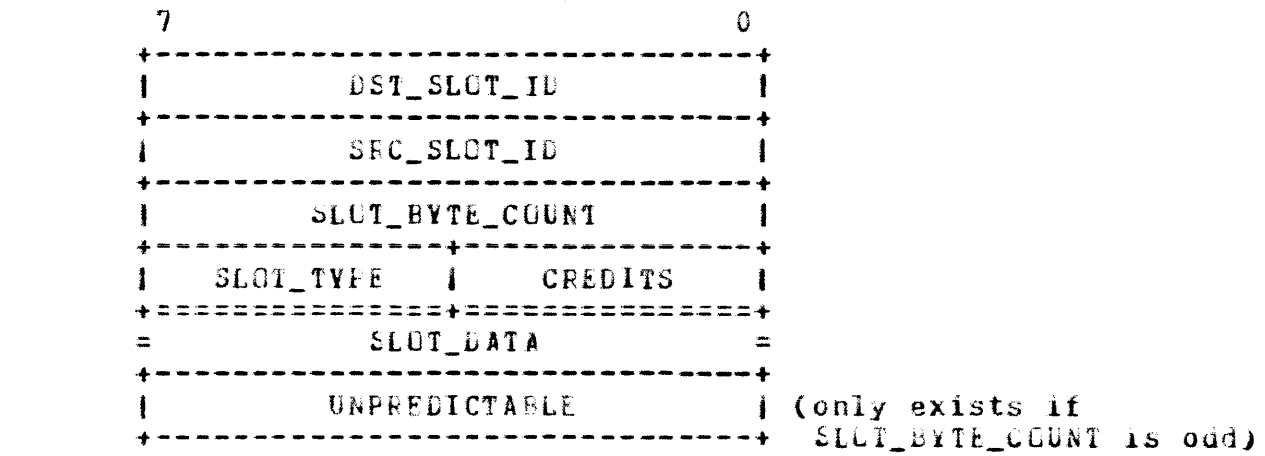
If a start slot is received (see slot state tables), the format of the slot is:



- 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 Data_a 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.

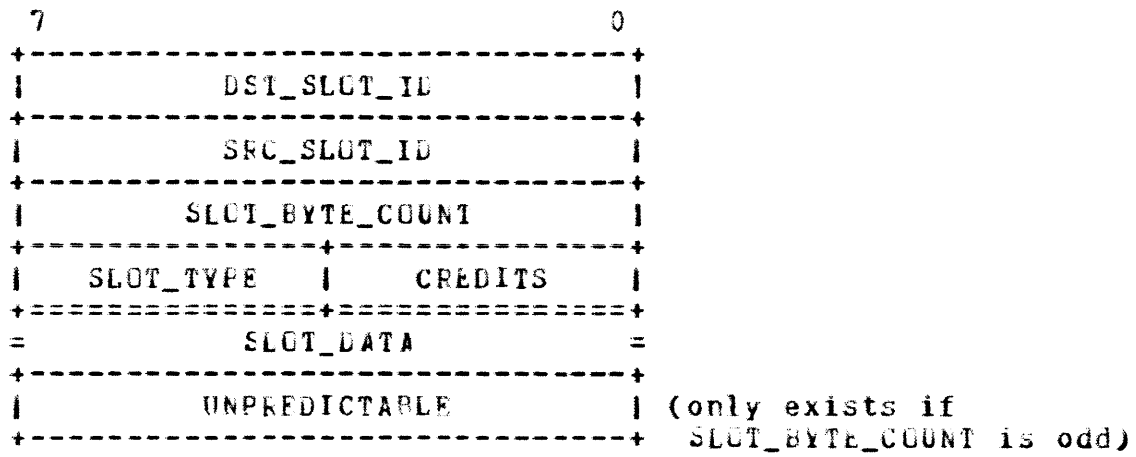
+ o DST_SLOT_NAME_LEN (1 byte unsigned) - The byte count of the next
+ field.
+ o DST_SLOT_NAME - The name of the host slot block.
+ o SRC_SLOT_NAME_LEN (1 byte unsigned) - The byte count of the next
+ field.
+ o SRC_SLOT_NAME - The name of the server slot block.
o STATUS - The remainder of the status field meanings are defined separately for each service class.

10.1.2.2 Data_a Slot -



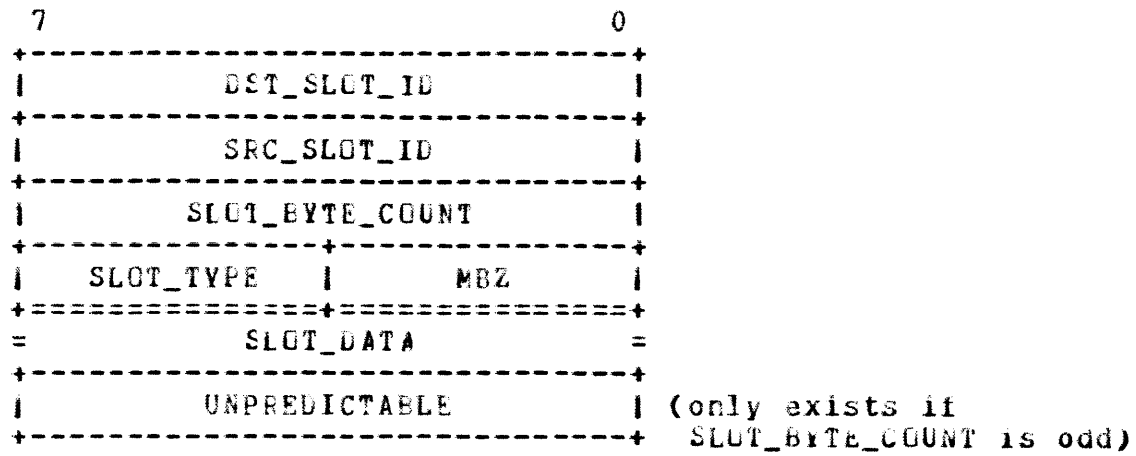
- DST_SLOT_ID - a reference to a slot block
- SRC_SLOT_ID - a reference to a slot block
- SLOT_BYTE_COUNT - an unsigned integer count of the length of the SLOT_DATA field.
- CREDITS (4 bits) - a 4-bit positive integer equal to the number of credits being transferred.
- SLOT_TYPE (4 bits) - the value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

10.1.2.3 Data_b Slot -



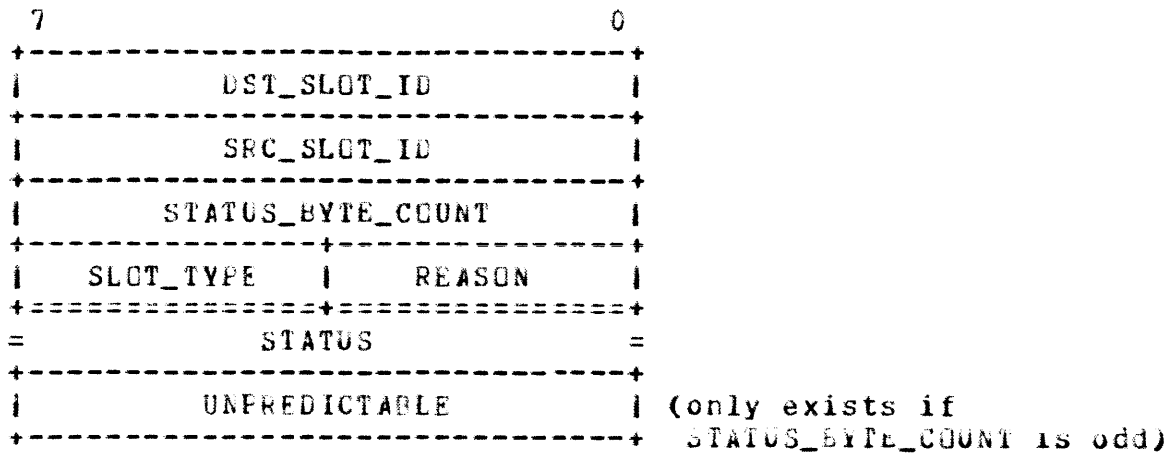
- DST_SLOT_ID - a reference to a slot block
- SRC_SLOT_ID - a reference to a slot block
- SLOT_BYTE_COUNT - an unsigned integer count of the length of the SLOT_DATA field.
- CREDITS (4 bits) - a 4-bit positive integer equal to the number of credits being transferred.
- SLOT_TYPE (4 bits) - the value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

10.1.2.4 Attention Slot -



- 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.

10.1.2.5 Reject Slot -



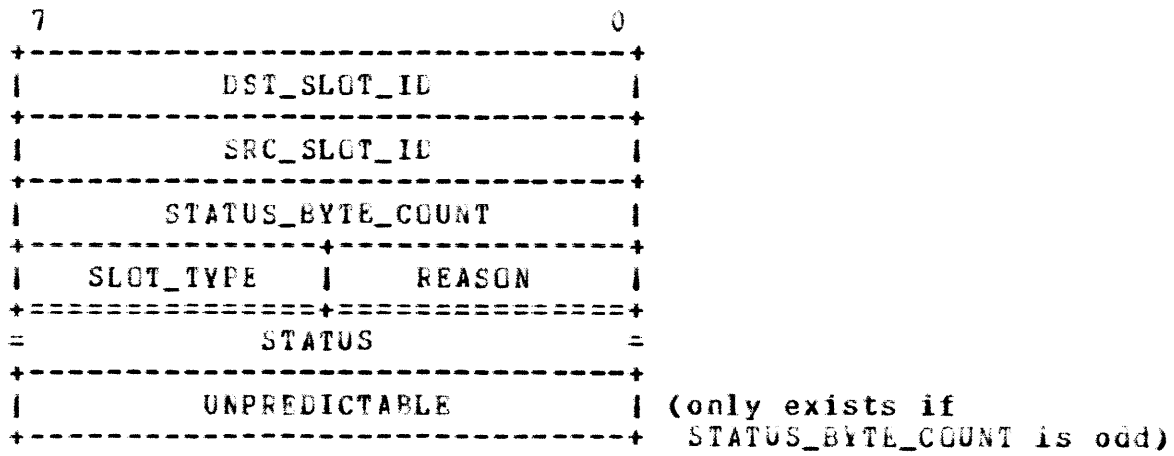
- o DST_SLOT_ID - a reference to a slot block
- o SRC_SLOT_ID - a reference to a slot block
- o STATUS_BYTE_COUNT - an unsigned integer count of the length of the STATUS field.
- o REASON - an unsigned 4-bit integer, one of the following reasons apply:

  1. user requested disconnect
  2. system shutdown in progress
  3. invalid slot received
  4. invalid service class
  5. invalid group code
  6. insufficient resources to satisfy request
  7. (invent your own, please get reasons added to this document)

- o SLOT_TYPE - the value 12. (1100)
- o 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.

10.1.2.6 Stop Slot -



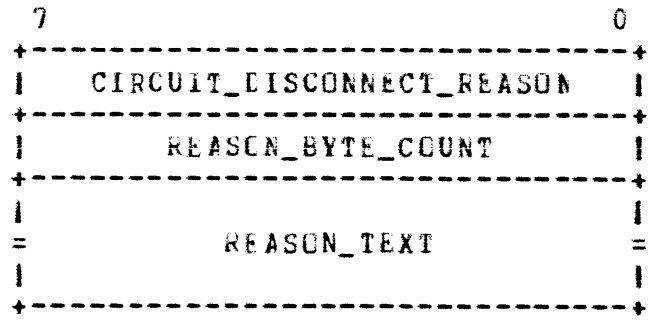
- DST_SLOT_ID - a reference to a slot block
- SRC_SLOT_ID - a reference to a slot block (must be zero)
- STATUS_BYTE_COUNT - an unsigned integer count of the length of the STATUS field.
- REASON - an unsigned 4-bit integer, one of the following reasons apply:

  1. user requested disconnect
  2. system shutdown in progress
  3. invalid slot received
  4. invalid service class
  5. invalid group code
  6. insufficient resources to satisfy request
  7. (invent your own, please get reasons added to this document)

- SLOT_TYPE - the value 13. (1101)
- STATUS - STATUS_BYTE_COUNT bytes of status. The status field meanings are defined separately for each service class.

10.1.3 Stop Message Format -

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



- CIRCUIT_DISCONNECT_REASON (2 bytes unsigned) - A zero value means no reason is given. The currently defined reasons are:

  1. No slots connected on virtual circuit.
  2. Illegal message format received.
  3. VC_halt from user.
  4. No progress is being made.
  5. Time limit expired.
  6. LAT_MESSAGE_RETRANSMIT_LIMIT reached.
  7. Insufficient resources to satisfy request.
  8. SERVER_CIRCUIT_TIMER out of desired range.
  9. Unknown PRODUCT_TYPE_CODE in Start message.
  10. Unsupported PRODUCT_TYPE_CODE in Start message.
  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.

11 ISSUES FOR FURTHER STUDY

1. multicast incarnation value meaning and format
2. protocol "correctness" will be proved by field test
3. performance measurements need to be done
4. desireable for terminal server to implement transmit timer/host
5. round-robin for terminal server host selection by name
6. 7-bit vs 8-bit ASCII
7. stopping service
8. stop slots mapping same as run slot
9. should start->run transition generrate a run message
10. who allocates start slots
11. patching

12 REFERENCES

- "The Ethernet - A Local Area Network - Data Link Layer and Physical Layer Specifications", DEC-INTEL-XEROX, V2.0, September 30, 1980.
- DNA NI DATA LINK ARCHITECTURAL SPECIFICATION, Bob Stewart, Tony Lauck, Digital Equipment Corporation, 1982.
- DNA ETHERNET NODE PRODUCT SPECIFICATION, Tony Lauck, Digital Equipment Corporation, 1982

APPENDIX A

SERVICE CLASS 1 - INTERACTIVE TERMINALS

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

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

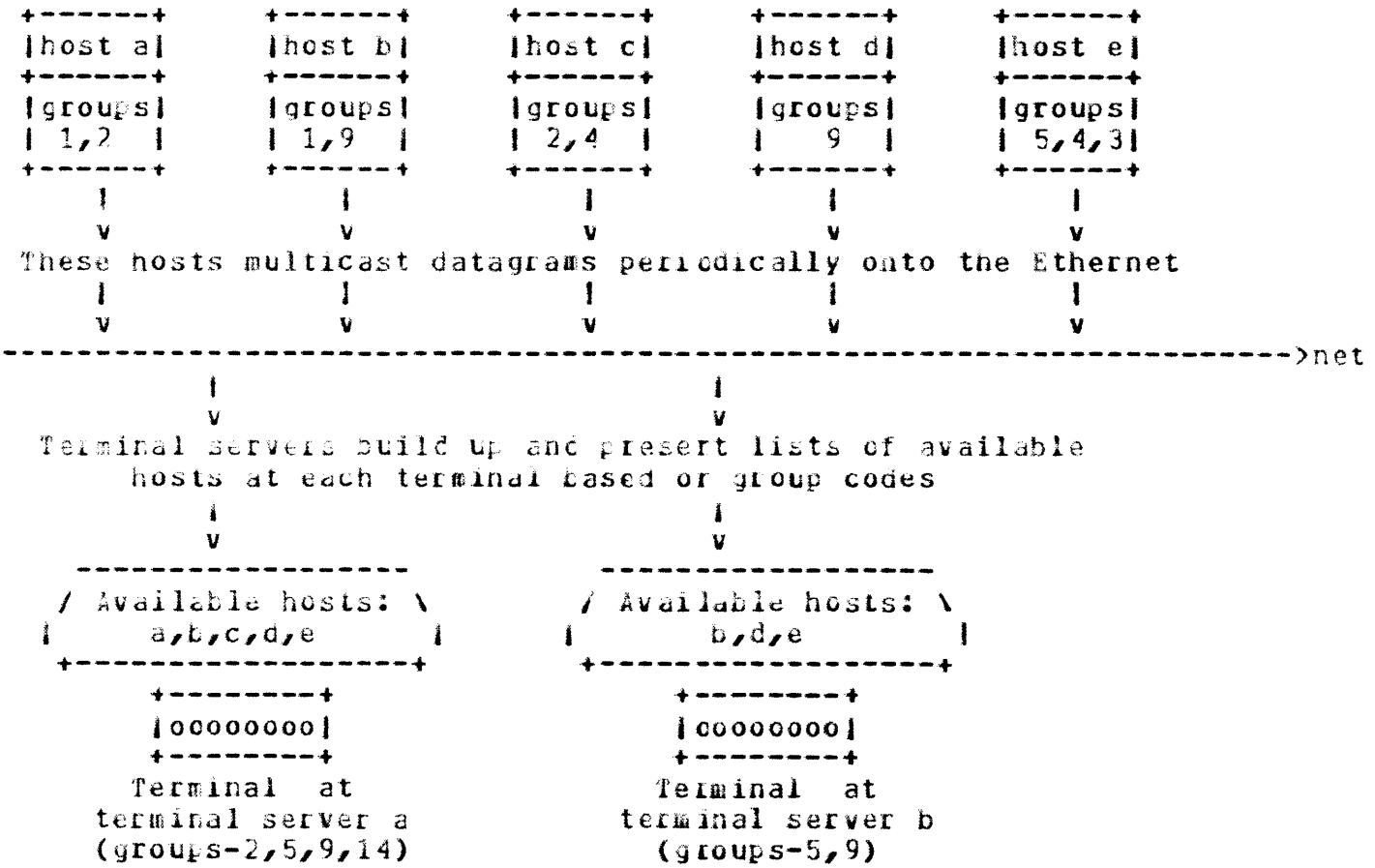
The directory service is very responsive to sudden changes in the local area topology. All terminal servers discover that a particular host is available within milliseconds of the host announcing the service. If a host should crash, the terminal servers can notify the users within a few seconds that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

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

A.1.1 Groups

Connectivity may be restricted by use of group codes. This restriction allows segmentation of the computing resources based on such criteria as departmental ownership or physical location.

Hosts are assigned to one or more groups. Terminals (or terminal servers) are also assigned to one or more groups. Whenever a terminal and a host belong to the same group, those two system can interact. For example:



The HOST_GROUPS field in the multicast message is a bit mask of 256 bits. If a bit is set, then the host is a member of that group. The first bit of the mask (bit 0) corresponds to group 1. A maximum of 256 groups can be specified (1-256) and therefore this field's maximum length is 32 bytes.

The purpose of the host group field is to allow the user at a server to control the display, not to restrict connectivity between hosts and terminal servers. Without this capability, a server user might be annoyed by the length of the display of available SLOT_NAMES.

Terminal servers (users) are assigned group numbers and discard all multicast datagrams that do not specify one of the group numbers assigned to the terminal server.

When group only group zero is enabled in the terminal server, only multicast messages which do not specify any group codes (i.e., specify the HOST_GROUP_LENGTH field as zero) are displayed to the users.

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

A.1.2 Host

A.1.2.1 Initialization -

A host system manager may specify:

- group codes (R)
- host 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
- 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. In order 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 that has no assigned CIRCUIT_NAME should not announce start of service. Instead, an error should be returned to the system manager.

A.1.2.2 Host Group Codes -

If a host wishes to allow any terminal server to attempt to utilize services available at the host, then no group numbers should be assigned at initialization. If a host wish to allow it's services to be available to only certain terminal server (users), then the coordination of a facility-wide group codes must be coordinated by the facility manager.

A.1.2.3 Host Names -

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

Although an ASCII_NAME 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 must specify at least one CIRCUIT_NAME in the multicast message.

A host can specify more than eight SLOT_NAMEs, a terminal server is required to support a minimum of eight SLOT_NAMEs. A terminal service is require to update all of the information in a multicast datagram, or none of

A.1.2.4 Name Order In Multicast Message -

The ordering of names in the multicat message is not arbitrary. The first name specified must be a CIRCUIT_NAME. This name cannot be ommitted.

Following the first CIRCUIT_NAME, one or more SLOT_NAMEs can be specified. This list of SLOT_NAMEs is always bound to the CIRCUIT_NAME which immediately preceeds it.

CIRCUIT_NAME(S) and associated SLOT_NAME lists can then be repeated up to a limit of 255 names.

For an example of this, see the section titled LOCAL AREA DIRECTORY SERVICE.

A.1.2.5 Steady-state Operation -

The host 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.1.2.6 System Shutdown -

When a host is "shutting down", the HOST_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.1.3 Terminal Server

A.1.3.1 Initialization -

Whenever a terminal server is initialized:

- Group code 0 is enabled (all groups are displayed).
- ^S and ^Q are used as the output flow control characters
- ^Q and null (or ^S and ^Q) are used as the input flow control characters
- These same flow control defaults are used when a slot session is established. Of course the host might change these values. When the terminal enters "local" mode, the original default flow control values are restored. If a previously active session is resumed, the flow control setting of the session must be restored.

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 operating systems 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 assigned with a single command.

If a privileged user disables group code 0 in the Terminal Server, and no multicast messages has yet been received that specifies one of the group codes 1-256, then no SLOT_NAMEs will be displayed to the user.

A.1.3.2 Building The Circuit Name Database -

Each Terminal Server build 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 none of it.

Terminal servers receive multicast datagrams periodically from each host. 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). If only group code 0 is enabled in the terminal server, then only multicast messages specifying the HOST_GROUP_LENGTH field as zero are processed.

If the multicast message contains a CIRCUIT_NAME which is not already in the list, a new circuit entry is created and the information in the multicast datagram is parsed into the entry. If the CIRCUIT_NAME has associated SLOT_NAMES, these too are added to the entry. This is repeated for each CIRCUIT_NAME in the multicast message.

If the CIRCUIT_NAME is already entered (with the same Ethernet address), 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. If this field has changed, then the information in the multicast datagram is reparsed into the list entry corresponding to the CIRCUIT_NAME. The CHANGE_FLAGS field should be used to reduce the CPU time necessary to parse the entry.

If the CIRCUIT_NAME is already entered in the list, but with a different Ethernet address, DUPLICATE_CIRCUIT_NAME counter should be incremented. Then the list entry should be created for this CIRCUIT_NAME as described above.

Servers maintain the list of all names in memory.

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

1. Unreachable hosts - these are hosts that are known to be unreachable because the LAT_MESSAGE_RETRANSMIT_LIMIT was reached on an active virtual circuit associated with the CIRCUIT_NAME.
2. Unknown hosts - these are hosts that 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 hosts that have indicated that they are no longer accepting connections.
4. Reachable - these are hosts that are reachable, but no virtual circuit is currently established to the CIRCUIT_NAME.

If after all of the above entries have been purged from the CIRCUIT_NAME database, no entries are available, the multicast message is discarded.

A.1.3.2.1 Error Recovery -

If a connect to a SLOT_NAME name fails, the list of SLOT_NAMEs is searched for an alternate CIRCUIT_NAME path to the SLOT_NAME. If one is found, a connection is attempted. This continues until a connection succeeds or until possible CIRCUIT_NAME paths to the SLOT_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 is purged from the list of available hosts displayed to the terminal server user.

A.2 IMPLEMENTATION ISSUES

A number of very different host services can be presented using the multicast message defined by service class 1.

A.2.1 Multiple Service Access Points

Define a service access point (SAP) as a preallocated host slot block in the Halted state with an associated SLOT_NAME. For example, consider this host topology: eight physical RS232 ports and a single Ethernet port. The problem is how to allow users to select a specific host SAP rather than any random SAP in the host.

This can be accomplished by giving each different host SAP a different SLOT_NAME in the multicast message transmitted by the host system. This will cause the SLOT_NAMES names to be presented to the users seperately, but all sessions will be piggybacked over a single virtual circuit.

A.2.2 Cluster Static Load Balancing

Clusters of machines might choose to present the same SLOT_NAME in their multiple multicast messages if they offer equivalent services. By cooperating among themselves to establish a common SLOT_NAME with appropriate SLOT_RATINGS, the cluster members canarrange to share the terminal user load. Digital Equipment VaxCluster present this type of name space to LAT terminal servers.

A.2.3 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 processor.

This can be accomplished by assigning multiple CIRCUIT_NAMEs in a single multicast message transmitted by the host Ethernet port. A similar effect can be accomplished by transmitting multiple Ethernet multicast messages, each specifying a different CIRCUIT_NAME, however this second method consumes more Ethernet bandwidth and processor time. This will cause a terminal server to establish a new virtual circuit for each different CIRCUIT_NAME.

Each CIRCUIT_NAME can still specify one or more SLOT_NAMES. This would allow piggybacking of sessions as usual. The smae SLOT_NAME can be assigned to more than one CIRCUIT_NAME to achieve static load balancing.

A.3 SERVICE CLASS 1 MESSAGE FORMAT EXTENSIONS

Each service class can define extensions to the messages in the main body of the document. Service class 1 extends the Start slot and defines special meanings to the Attention slot.

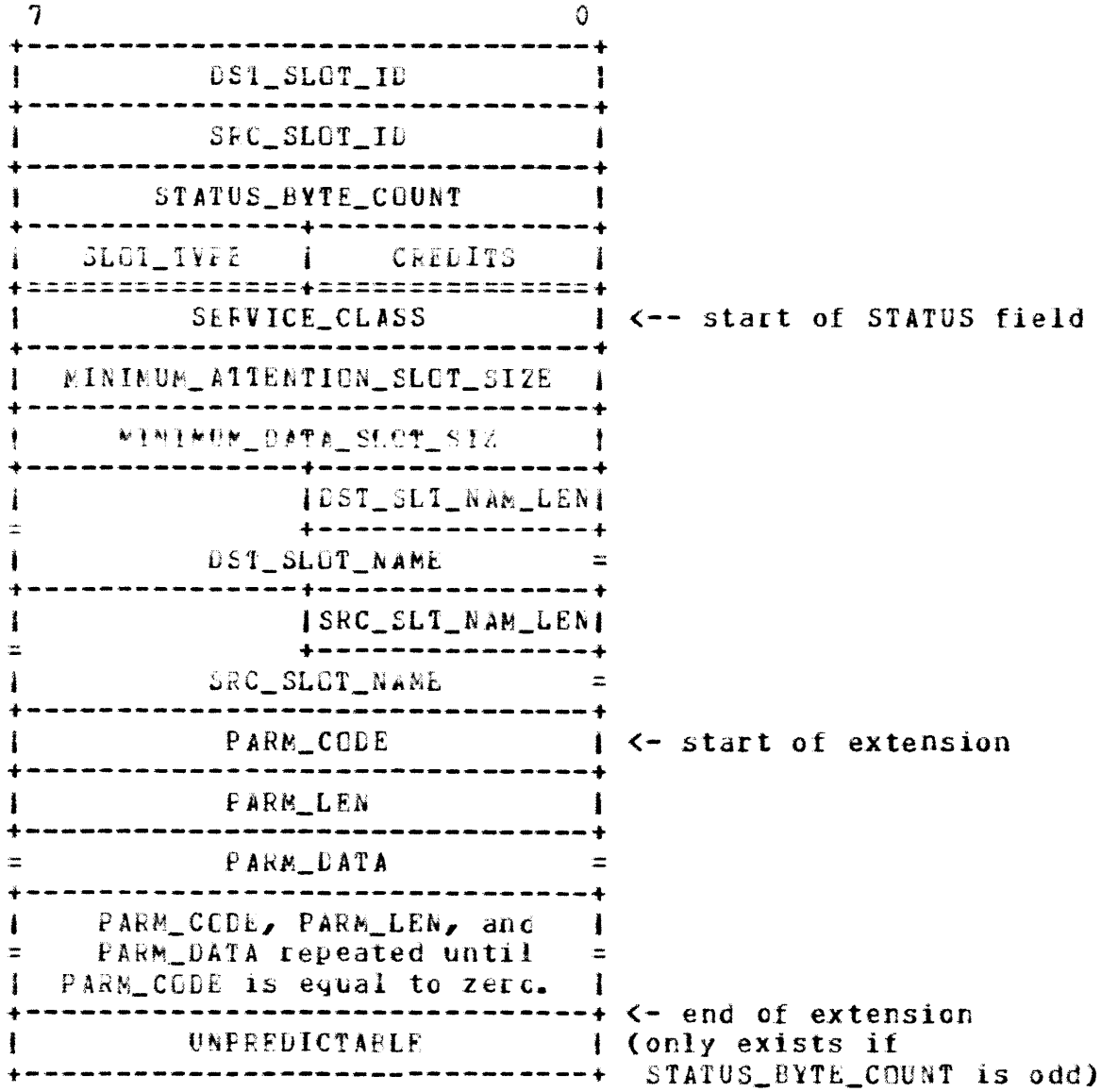
If the slot byte count is in conflict with a field byte count, the slot is invalid. If the slot byte count truncates an extension to the slot, the slot is valid and the extension is not supplied. If a byte counted field within a slot status field is specified as zero length, the next byte following the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

Bits are transmitted 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.3.1 Start Slot Status Field

The Start slot sent to the host by the terminal concentrator is extended by this service class. The Start slot sent by the host is not extended. All of the fields in the Start slot extension are optional. The format of the server-to-host Start slot is:



- DST_SLOT_ID - a handle on the remote slot block
- SRC_SLOT_ID - a handle on the local slot
- ...
- PARM_CODE (1 byte) - A parameter code. 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.

The following parameters are defined for service class 1 - interactive terminals:

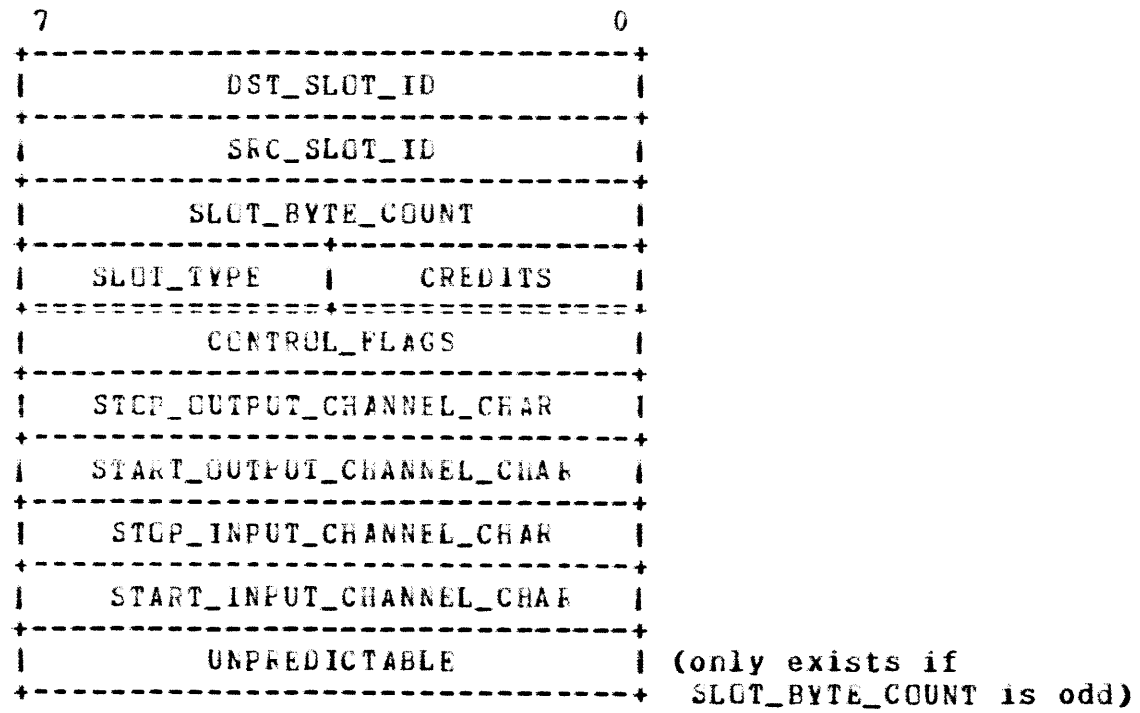
- Parameter code 0 is reserved.
- Parameter code 1, parameter length = 2 - Flag word.
  - Bit 0 - Remote (modem) line if set to 1.
  - Bit 1 - Secure line if set to 1. Server will abort session if <break> condition is detected.
  - Bits 2-15 are unpredictable.
- Parameter codes 2 through 127 are unpredictable.

A.3.2 Data_b Slot Status Field

The Data_b slot is extended by this service class. Although the slot can be sent by either the host or the terminals server, the host optionally implements the break and flow control functions. The terminal server always ignores the break flag. The terminal server must implement the flow control and abort functions.

Note that this slot is flow controlled.

The format of the Data_b slot is:



- 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.
- CREDITS (4 bits) - a 4-bit positive integer equal to the number of credits being extended.
- SLOT_TYPE (4 bits) - the value 11.
- CONTROL_FLAGS (8 bits) :
  - (bit 0) - Enable recognition of input flow control characters. If set, 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 terminal output stream as indicated, and discard the character.

- (bit 1) - Disable recognition of input flow control characters. If set, all characters in the terminal input stream are passed directly through to the host without interpretation.
- (bit 2) - Enable recognition of output flow control characters. If set, 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 buffering, 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 3) - Disable recognition of output flow control characters. If set, all terminal output characters are passed directly to the output stream. No characters are generated by the terminal server.
- (bit 4) - Break condition detected (terminal server to host only).
- (bit 5 through bit 7) - Unpredictable.

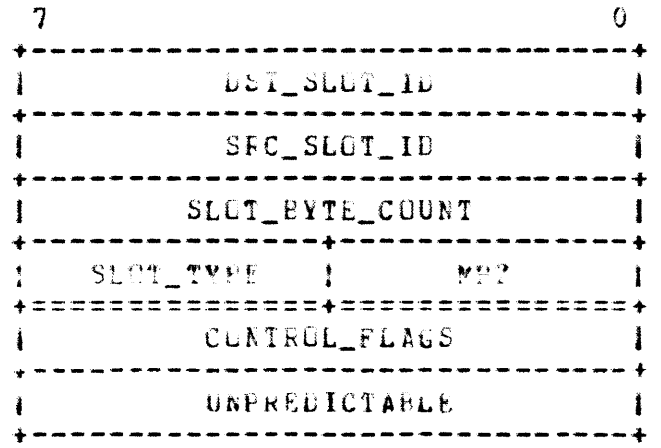
- STOP_OUTPUT_CHANNEL_CHAR - The value assigned to stop the terminal output stream if input flow control characters are enabled. The value assigned is normally control-S.
- START_OUTPUT_CHANNEL_CHAR - The value assigned to start the output channel if input flow control characters are enabled. The value assigned is normally control-Q.
- STOP_INPUT_CHANNEL_CHAR - The value assigned to stop the terminal input channel if output flow control characters are enabled. The value ssigned is normally control-S or control-G.
- START_INPUT_CHANNEL_CHAR - The value assigned to start the terminal input channel if output flow control characters are enabled. The value assigned is normally control-G or null.

A.3.3 Attention Slot Status Field

The Attention slot is extended by this service class. 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 format of the Attention slot is:



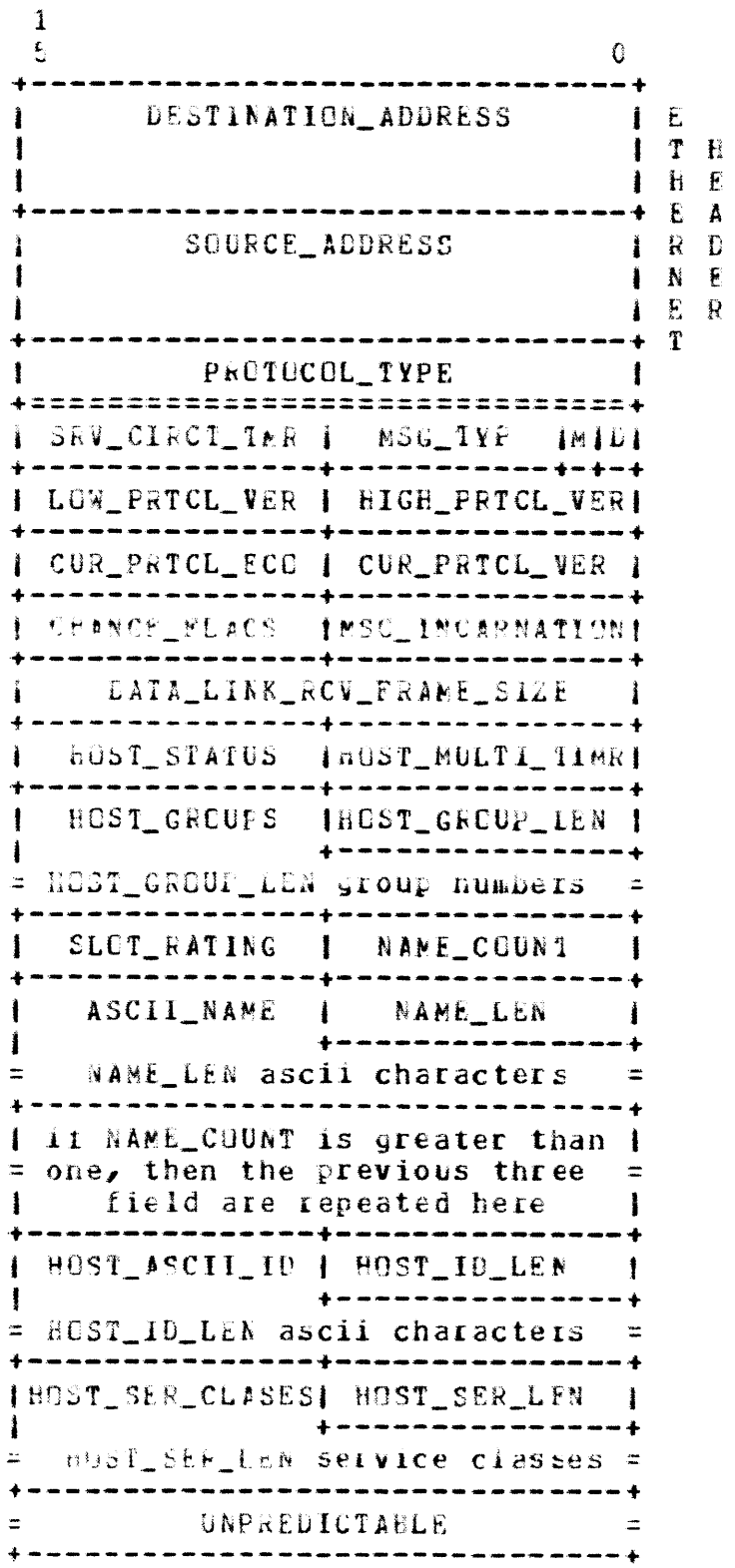
- DST_SLOT_ID - a handle on the remote slot block
- SRC_SLOT_ID - a handle on the local slot block
- SLOT_BYTE_COUNT - an unsigned integer count of the length of the SLOT_DATA field.
- MBZ (4 bits) - must be zero
- SLOT_TYPE (4 bits) - the value 11.
- CONTROL_FLAGS (8 bits) :
    - (bit 0 through bit 4) - Unpredictable.
    - (bit 5) - Abort. Causes buffered output data pipe to be flushed of all characters.
    - (bit 6 and bit 7) - Unpredictable.

A.4 MESSAGE FORMATS

This service class defines the following additional messages.

A.4.1 Patch Message

A.4.2 Multicast Datagram



- MSG_TYP (6 bits) - fixed at 40. (The value of the byte is 40 since the M and D flags are clear always clear.)

- SERVER_CIRCUIT_TIMER (1 byte) - Desired value in milliseconds. The host 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 host.

- LOW_PRTCL_VER (1 byte) - Lowest protocol version supported by host.

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

- CUR_PRTCL_ECO (1 byte) - ECO level of CUR_PRTCL_VER (this message). ECO (Engineering Change Orders) are always backwards compatible for any given protocol version. If an ECO makes a change incompatible with the previous version, a new (higher) protocol version must be allocated.

- MSG_INC (1 byte) - Message incarnation. This multicast datagram is transmitted periodically by each host. Any time ANY field within this message changes value relative to the previous message, the MSG_INCARNATION must be incremented by one. When the host 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 - host name(s) changed
  - bit 1 - host rating(s) changed
  - bit 2 - HOST_ASCII_ID changed
  - bit 3 - HOST_GROUPS changed
  - bit 4 - HOST_SER_CLASSES changed
  - bit 5 - HOST_STATUS changed
  - bit 7 - Remaining parameters changed (Server timer, multicast timer, Protocol version, ECO level or Data link frame size).

  If a field changes, the bit value toggles (0 -> 1 or 1 -> 0) and then remains at that value until as multicast messages are transmitted until the field changes again.

- DATA_LINK_RCV_FRAME_SIZE (2 bytes) - The minimum and maximum sizes of frames are restricted by the Ethernet specification to be in the range 46 through 1518 bytes. A host implementation must specify the guaranteed minimum size receive buffer that it queues to the data link layer of ethernet. Legal values are restricted by the LAT architecture to be in the range of 576 through 1518 bytes. Terminal servers never transmit a datagram to a host that is longer than DATA_LINK_RCV_FRAME_SIZE bytes. Hosts never queue a receive buffer smaller than this size.

- HOST_MULTICAST_TIMER (1 byte unsigned) - The minimum rate at which the host will send multicast messages in seconds.

- HOST_STATUS (1 byte bit mask) - Host status flags byte.

  - Bit 0 - Set to 1 if the host is not accepting new sessions.
  - Bits 1 through 7 - unpredictable.

- HOST_GROUP_LEN (1 byte unsigned) - A byte count of the HOST_GROUP field. A value of zero is legal and indicates that the host belongs only to group zero.

- HOST_GROUPS (HOST_GROUP_LEN bytes) - This field is specified as a bit-mask or 256 bits. A bit set to 1 indicates the host belongs to that group. The first bit of the mask (bit 0) corresponds to group 1.

- NAME_COUNT (1 byte unsigned) - This field is equal to the number of names offered. The next three fields are repeated NAME_COUNT times.

- SLOT_RATING (1 byte unsigned) - the rating of the associated name. If the name is a CIRCUIT_NAME, by convention, the value is equal to the number of interactive users on the host. If the associated name is a SLOT_NAME, the value is used to arbitrate between multiple CIRCUIT_NAME paths to the same SLOT_NAME.

+ - NAME_LEN (1 byte signed) - A byte count of the ASCII_NAME field. A value of zero is illegal. Positive counts are SLOT_NAMES. Negative counts are CIRCUIT_NAMES.

- ASCII_NAME (NAME_LEN bytes) - Any character in the ASCII_NAME field must be in the range 33 to 126 or 161 to 254 (this eliminates control characters). Lower case characters will be upcased before comparison.

- HOST_ID_LEN - (1 byte unsigned) - A byte count of the HOST_ID field. A value of zero indicates that no HOST_ID is available.

- HOST_ID (HOST_ID_LEN bytes) - An ASCII string of characters that will help the terminal server user confirm that the SLOT_NAME is really the service he expects. The host should not load control characters into this field. Terminal servers must support a minimum HOST_ID length of 64 characters.

- HOST_SER_LEN (1 byte unsigned) - A byte count of the HOST_SER_CLASES field. A value of zero is illegal.

- HOST_SER_CLASES (HOST_SER_LEN bytes) - A host might simultaneously offer multiple services. 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. The service classes

defined at present are:

- o CLASS 1 - Interactive terminals
- o CLASS 2 - Application terminals

APPENDIX B

SERVICE CLASS 2 - APPLICATION TERMINALS

B.1 INTRODUCTION

Service class 2 is not fully specified and has not been approved for implementation. An outline appears here simpy to show how service classes will be interrelated.

An Interactive terminal is a device under the control of a user, while an Application terminal is a device under the control of a computer process. In some cases an Application terminal may not even have a keyboard: line printers, video monitors and display windows fall into this class of Application Terminal.

Before a computer process can allocate an Application terminal and write to it, the name of the Application terminal must be made available to the computer process and the computer process must have the authority to use the device.

B.2 RELATIONSHIP TO SERVICE CLASS 1

Service Class 2 supports every detail of Service Class 1. The only difference between the two different service classes are the extensions to Service Class 1 made by Service Class 2.

B.3 ARCHITECTURAL MODEL

B.3.1 Host System As Initiator Of Connect

In order to:

- preserve the investment in the host implementation
- to allow the shared Application terminal to be arbitrated
- to allow the host system to initiate sessions to terminal server ports

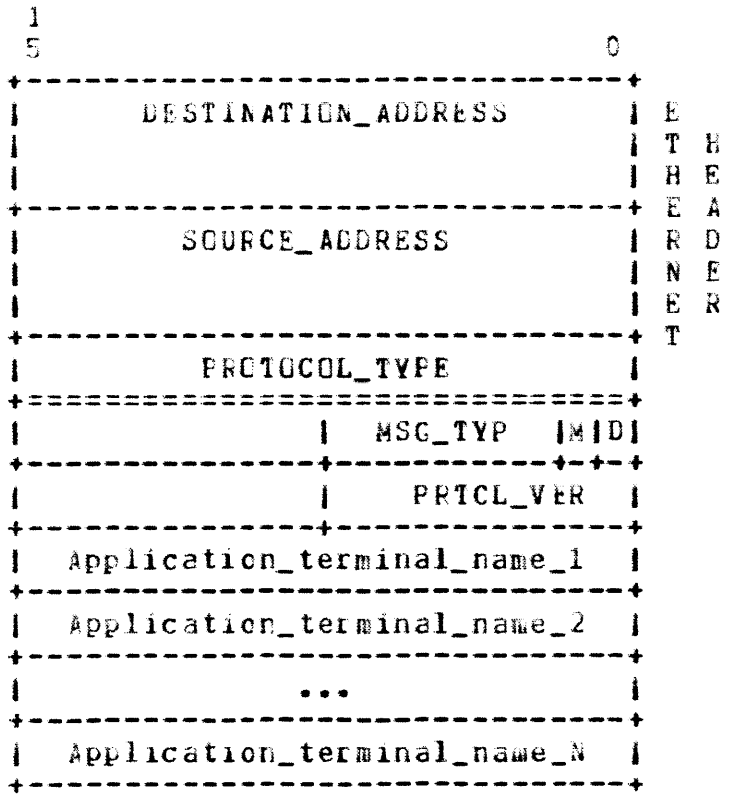
host application processes "solicit" sessions with Application terminals.

B.3.2 Authorization

In the model presented in this Service Class, the device itself decides which computer processes are authorized to allocate and use the device. A host computer process, no matter how privileged within the context of the host operating system, cannot allocated and use a Service Class 2 device unless the device has been set up to service the group to which the host system belongs.

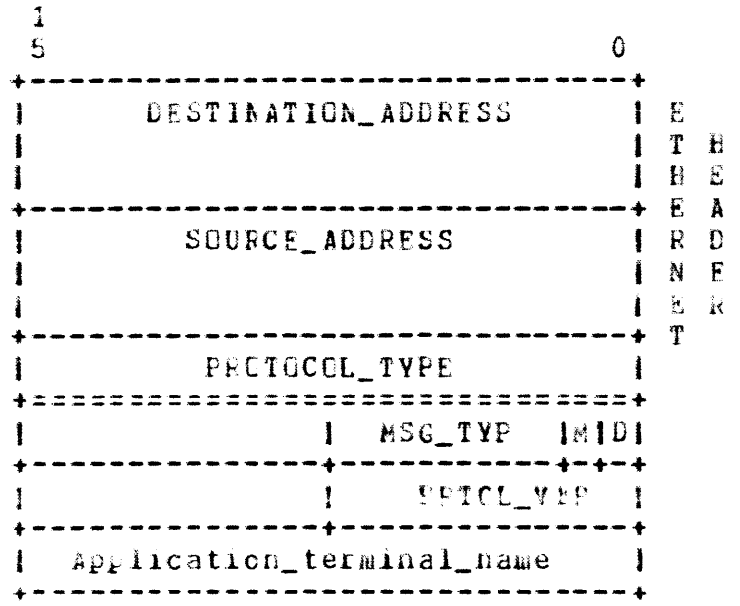
B.3.3 Advertising Application Terminals

Host system processes find out about Application devices by listening to to the Application terminal multicast address. Each server system supporting Application Terminals multicasts this message periodically. The message lists the names of the names of the Application terminals connected to the server system.



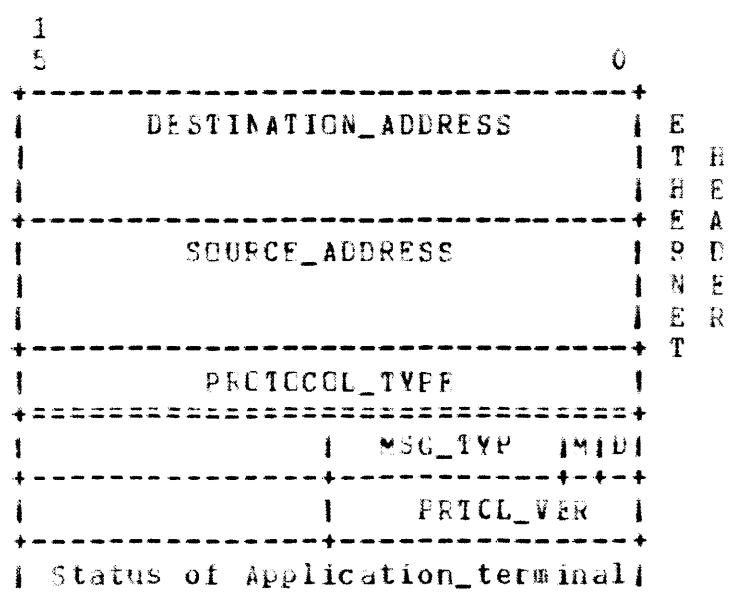
B.3.4 Soliciting Application Terminals

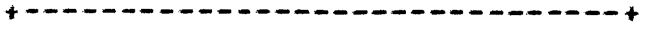
If a host system wishes to use an Application terminal, it must transmit this message to the server system on which the Application terminal is attached. This message is physically addressed.



B.3.5 Sharing

Since a device might be busy at the time service is requested by a host process, the following physically addressed message is used to reply to the host system requesting service:





INDEX

[End of Document]