Library -1

Bolt Beranek and Newman Inc.

Set II

REPORT 5256

LOGICAL ADDRESSING IMPLEMENTATION SPECIFICATION

May 1983

Submitted to:

Defense Communications Agency
Defense Data Network Project Management Office
Attn: Code B645
Washington, D.C. 20305

Submitted by:

Bolt Beranek and Newman Inc.
1300 North 17th Street
Arlington, VA 22209

Author: Andrew G. Malis

Contract No. DCA100-82-C-0076

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 5256 / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) LOGICAL ADDRESSING IMPLEMENTATION SPECIFICATION	5. TYPE OF REPORT & PERIOD COVERED FINAL
	6. PERFORMING ORG. REPORT NUMBER 5256
7. AUTHOR(S) Andrew G. Malis	8. CONTRACT OR GRANT NUMBER(S) DCA100-82-C-0076
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 10 Moulton Street Cambridge, MA 02238	10. PROGRAM ELEMENT PROJECT TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Bolt Beranek and Newman Inc. 1300 North 17th Street Arlington, VA 22209	12. REPORT DATE May 1983
	13. NUMBER OF PAGES 52
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Defense Communications Agency Defense Data Network Project Mgmt. Office Attn: Code B645 Washington, D.C. 20305	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

20 copies to DCA Code B645

2 copies to DCA Code C742	2 copies to DCA Code C440
2 copies to DCA Code C420	2 copies to DCA Code C402
2 copies to DCA Code C430	2 copies to DCA Code N260

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

PACKET SWITCHING NETWORKS, ARPANET, WWMCCS, NETWORK ADDRESSING

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report contains the design description for the software which implements logical addressing in the ARPANET packet switching system.

REPORT 5256

LOGICAL ADDRESSING IMPLEMENTATION SPECIFICATION

May 1983

Submitted to:

Defense Communications Agency
Defense Data Network Project Management Office
Attn: Code B645
Washington, D.C. 20305

Submitted by:

Bolt Beranek and Newman Inc.
1300 North 17th Street
Arlington, VA 22209

Author: Andrew G. Malis

Contract No. DCA100-82-C-0076

Table of Contents

FIGURES

1 Introduction

This is the specification for the upcoming implementation of the logical addressing (1822L) IMP, as functionally described in BBN Report 5255, _The ARPANET 1822L Host Access Protocol_. This specification includes the new data structures, IMP-to-IMP messages, and IMP algorithms that will be added to the 1822L IMP. It also specifies the implementation of the logical address translation database maintenance from NU.

A note on terminology: _name_ refers to an 1822L-style logical address, as defined in BBN Report 5255; _1822 address_ refers to 24-bit addresses used in 1822 leaders; _physical address_ refers to an actual host port on the network; and _1822L address_ refers to the 16-bit mapping used to refer to physical addresses in 1822L leaders. See section 2.1 of Report 5255 for further details.

There are a number of assumptions and general points that need to be listed before the specifics of the implementation are discussed:

o 1822L is not a package. Rather, it is an integral part of the NMFS IMP, much like multi-trunking and 128 channels. It simply requires too many changes to the existing IMP code to be a package, and incorporating it into the IMP also makes it simpler to use operationally.

- The 1822L database design is targeted for the efficient use of a database of up to about 2000 names, with an average of about 4 addresses per name. With a database of up to this size, all of the IMPs contain the full translation database, as was recommended in Report 4473, ARPANET Routing Algorithm Improvements - Volume I. This database sits in stolen IMP buffers, and is downline-loadable with the rest of the IMP code in a reload. The database itself, and its updates, are sequenced for consistency checking. No separate cache for frequent translations exists per se, but the transmit message blocks serve as an effective translation cache for connections that are already open. If the database were to grow beyond these bounds, the database routines are highly modularized to allow easy replacement with another set of routines that would use an external database server with a large internal database cache with entry aging. Also, if the use of uncontrolled messages were to grow, a separate uncontrolled message translation cache could also be added to augment the translation cache already provided by the connection blocks for regular messages.

- A flag in each IMP specifies whether logical addressing is in use. If this flag is off, the IMPs do not require any space to be reserved for the translation database. Even if this bit is off, the IMPs accept 1822L leaders (containing 1822L addresses). Of course, 1822L leaders using names to identify either the source or destination host will not be accepted, since there would be no way to translate the name, and such messages will be rejected.

o With one exception, all translations take place in the source IMP, with no tandem IMP translation. The one exception is if a tandem IMP receives a logically-addressed uncontrolled packet, the originally translated destination is no longer reachable, and the name maps to another reachable destination, then the packet is re-addressed by TASK in the tandem IMP.

o Names used with end-to-end connections are translated only at the connection setup time. If a destination name maps to more than one physical address, multiple attempts are made to set up the connection, until either the connection has been made or the end of the physical address list is reached. If, during the lifetime of a connection, the destination IMP or host port goes down, the existing connection is destroyed and the source host receives a destination dead for any messages with outstanding RFNMs on that connection. However, the next time the host submits a message for that same destination name, the name is re-translated to set up a new connection.

Only if all of the physical ports to which the destination name maps are down will the source host be unable to get messages to that destination.

o Three different address selection criteria are available for the name mapping process. When translated, each name uses one of the three criteria, which is selected by the translation database entry for that name. The three criteria are:

- Attempt each translation in the order in which the physical addresses are listed in the translation database, to find the "first reachable" physical host address. This list is always searched from the top whenever an uncontrolled packet is to be sent or a connection has to be created. This criterion is the most commonly used.

- Selection of the "closest physical address", which uses the routing tables to find the translation to the destination IMP with the least delay path.

- Use "load leveling". This is similar to the second criterion, but differs in that searching the address list for a valid translation starts at the address following where the previous translation search ended. This attempts to spread out the load from any one IMP's hosts to the various host ports associated with a particular name. Note that this is NOT network-wide load leveling, which would require a distributed algorithm and tables.

o Maintenance of the translation database is handled from NU. The database updating procedure consists of update messages that are sent from NU to the IMPs, and are used for incremental changes to the database in the IMPs. The Logical Address Update Fake Host receives and processes the updates. There will also be procedures for loading a completely new database into an IMP in its loader/dumper, in order to introduce an initial logical addressing database to a network. Once one IMP has the database, reloading the other IMPs from this "seed" IMP, and then from each other, will spread the database throughout the network.

2 IMP Data Structures

Logical addressing requires a translation database, of course, and that database is described in this section. Logical addressing also requires changes in two existing IMP data structures, the receive and transmit blocks. These changes are also detailed below.

2.1 The Translation Database Format

The logical addressing translation database consists of two main parts, the NAME and the ADDR tables. In addition, an auxiliary table, the PORT table, is used to speed up certain types of translations. These are discussed in the following sections.

2.1.1 The NAME and ADDR Tables

The main portion of the translation database is split into two tables, the NAME and ADDR tables. The NAME table is made up of NAME entries, each of which contains a name and a pointer to a list of physical addresses to which the name maps. This list is kept in the ADDR table, and consists of a contiguous list of ADDR entries, each of which contains a physical address and some status bits.

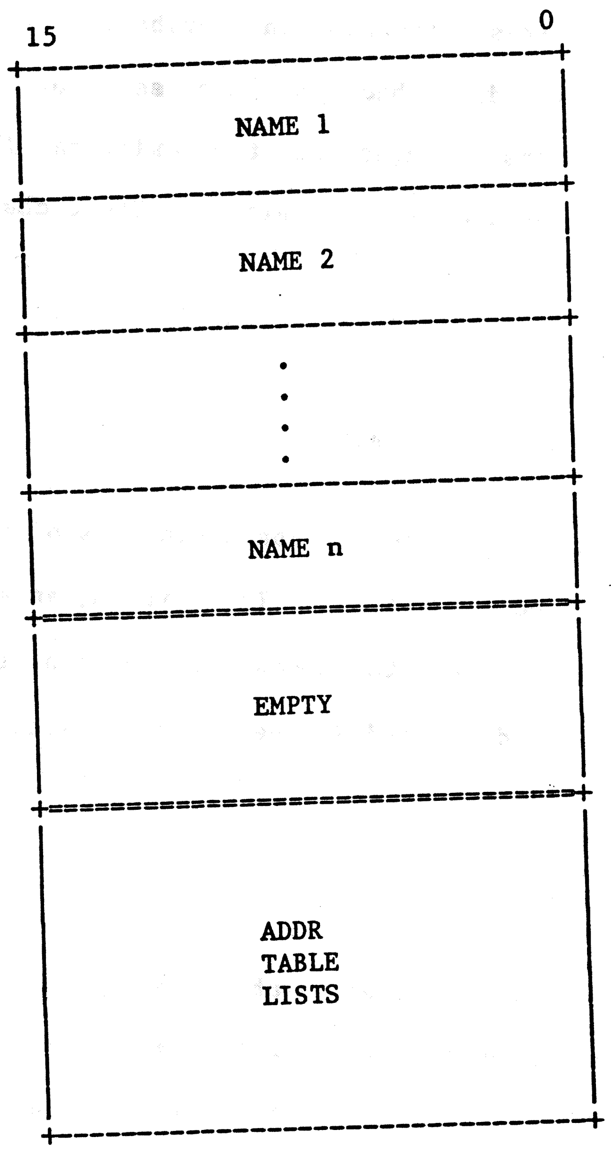

Figure 1. Logical Addressing Translation Database

Figures 1 - 3 show these tables in varying detail. In figure 1, the entire database is shown filling one large area in memory, which is an even multiple of buffers in length (and actually sits in stolen buffers). The NAME table portion of the database starts at the top and grows down, and the ADDR table starts at the bottom and grows up. In the middle is an empty space for table expansion. The NAME table entries are always kept sorted and contiguous, but the ADDR table lists are placed randomly and can be separated by empty space, although each list's entries are contiguous.

```
 15                              0
+--------------------------------+
|                                |
|           1822L NAME           |
|                                |
+--+--+--------------------------+
|C |G |                          |
|R |R |    ADDR LIST POINTER     |
|I |P |                          |
+--+--+--------------------------+
```

Figure 2. Name Entry

Figure 2 shows a NAME entry in more detail. The first word contains the name, which will be in the range 40000 - 177777 octal (remember, the range 0 - 37777 is reserved for 1822L addresses). The second word contains two bits of selection criterion, one bit that identifies the name as being a group address rather than a name, and 13 bits that contain the entry number of the beginning of the ADDR list for this NAME entry. The 13 bits allow a maximum of 8,192 entries in the ADDR table (potentially filling 16K words). The selection criterion values are:

00: Use the first reachable physical address
01: Use the closest physical address
10: Use load leveling
11: Unused

The group address bit is currently unused, but will be when group addressing is implemented. Group addressing will allow a single message to be delivered to every host in a group, and will also allow indirect names (names that resolve to other names as well as to physical addresses).

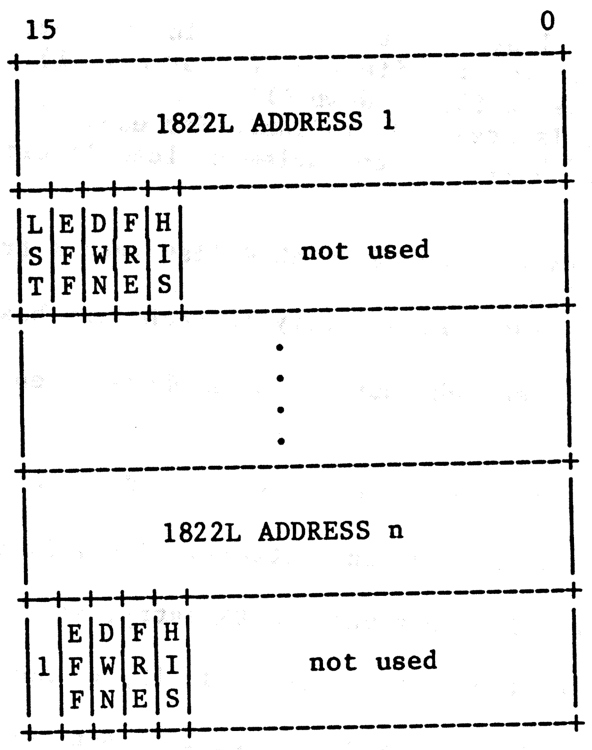

Figure 3. ADDR List

Figure 3 shows an ADDR list. It consists of one or more ADDR entries, each of which has two words. The first word of each ADDR entry contains an 1822L address, which identifies the physical host port to which this ADDR entry refers. In the future, when group addressing has been implemented, this word will hold 1822L names as well as addresses. The second word contains status bits in the high-order five bits, which are used as follows:

bit #
-----
15. This is the last ADDR table entry in the name list.
14. This translation is effective (0) or not (1).
13. This host is up (0) or down (1).
12. This entry is free and available for use.
11. A "history" bit, used to implement load leveling.

Bit 15 is used to find the end of an ADDR list when a translation is performed. The last ADDR entry in each list has this bit turned on, so that the search routine knows where to end.

Bits 14 and 13 are used to keep messages to dead and/or non-effective hosts from flooding the network. Every five minutes, these bits are cleared in every valid ADDR entry, and these bits are set if a destination host is down (bit 13) or non-effective (bit 14). Logically-addressed messages will be sent only if both bits are off.

Bit 12 assists the table space management needed to efficiently update the translation database. All of the unused ADDR list entries are marked with this bit for easy management and re-use. The first empty entry in a contiguous block of empty entries also has bit 15 turned on in its second word, and its first word contains the number of contiguous empty words in the block.

Bit 11 is used to implement the load leveling selection algorithm. This algorithm, further described in section 5.1, requires one bit of history in the ADDR table.

The IMP uses a block of variables to manage the above data structures. These variables include pointers to the start address of each table and to the start of the empty space between the tables. They also include words to hold the number of entries in each table, the size of the empty space, and the amount of free space in the ADDR table. As will be discussed in section 4, these variables are identical in all of the IMPs in a network so long as the IMPs contain consistent databases.

2.1.2 The PORT Table

BBN Report 4473 <u>ARPANET Routing Algorithm Improvements</u> - <u>Volume I</u> discusses the need to keep, for each local host port, a list of the names authorized for that port, primarily to speed up translations involving local host ports and the processing of Name Declaration Messages (NDMs) from the hosts. Every message from each local host has to have its source name checked for authorization and effectiveness as well as to insure that the source name really refers to the host port that sent the message.

Searching the database to perform these checks could slow down traffic from the hosts considerably. The normal method for searching the translation database is a binary search of the NAME table to find the matching name, and then, if one is looking to match that name with a particular host port, to linearly search the associated ADDR list to find the entry for the host port in question. The NAME table search needs to examine a number of names equal to the log base two of the number of names in the NAME table. If there are 1024 names, for example, then the search can be expected to require ten stabs into the table. This is obviously a lot of work to perform in order to check the source name on every message from the hosts.

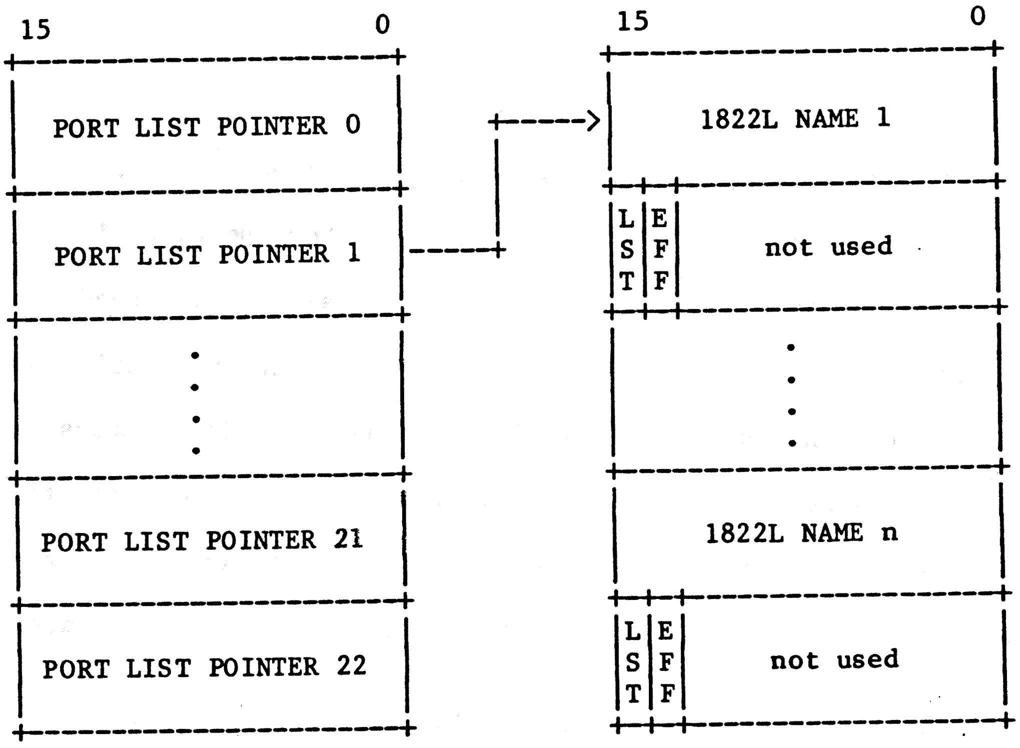

Figure 4. PORT Table and a PORT List

To make this check go faster, the PORT table lists, for each local host port, the names that translate to it. The table itself consists of twenty-three pointers, one per host port (sixteen real and seven fake hosts), to these lists of names (see figure 4, which shows the PORT table and the PORT list for local host port 1). Each PORT list is simply a contiguous set of name entries, with two words per entry: the first word contains an 1822L name that maps to the port, and the second word uses the two most significant bits:

bit #	
15.	This is the last name entry in the PORT list.
14.	This translation is effective (0) or not (1).

These two bits serve the same usage as the corresponding bits in the ADDR lists. Bit 14, the effective bit, will be identical in both the PORT table name entries and in their corresponding ADDR entries in the main database tables.

Every port has a PORT list with at least one PORT entry. If there are no names that refer to a certain port, then its list will consist of one dummy entry that has its 1822L name set to zero, which is an invalid 1822L name or address. On the other hand, if there are one or more names that refer to a port, then that port's list will contain one or more non-dummy entries that refer to the name or names. Using this dummy entry makes inserting new names and deleting names from the PORT lists much easier than if some of the PORT table pointers were null.

Of course, the PORT table is affected by updates to the database. Whenever the IMP is restarted or reloaded, new PORT lists are constructed from scratch by running through the entire database from top to bottom. If an update takes place, the added or removed ADDR entries are checked to see if they refer to local host ports, and if so the associated PORT list(s) are updated. However, PORT lists for unaffected ports stay intact.

The time the IMP puts into maintaining the PORT table is more than made up by the time saved during translations of local host addresses, which occur on every message from each host port.

2.2 Receive and Transmit Message Blocks

The only existing IMP data structures requiring any changes are the receive and transmit message blocks. When a connection is being established between 1822L hosts, the sending host provides its own and the destination's names. Both names have to be provided to the destination host when it receives messages on the connection. However, it would be wasteful to include this information in each message that traverses the network, and compatibility with the header formats used by those IMPs without logical addressing would be lost. Instead, the names only have to be sent once, in the GETBLK message.

```
 15                             0
+-------------------------------+
|          FOREIGN IMP          |
+---------------+---------------+
| FOREIGN HOST  |  LOCAL HOST   |
+-+------+------+---------------+
|P| HAND | FUSE | FOREIGN BLOCK |
+-+------+------+-------+-------+
|     TMESS     | LUSE  |  AGE  |
+-+-+-+-----+---+-------+-------+
|R|I|S|ALLOC|INC| MESSAGE BITS  |
+-+-+-+-----+---+---------------+
|         LOCAL NAME *          |
+-------------------------------+
|        FOREIGN NAME *         |
+-------------------------------+
|   HASH BUCKET NEXT POINTER *  |
+-------------------------------+
```

Transmit Message Block

```
+-------------------------------+
|          FOREIGN IMP          |
+---------------+---------------+
| FOREIGN HOST  |  LOCAL HOST   |
+-+------+------+---------------+
|P| HAND | FUSE | FOREIGN BLOCK |
+-+------+------+-------+-------+
|     RMESS     | LUSE  |  AGE  |
+---------------+-------+-------+
|            RSTATE             |
+-------------------------------+
|             RTYPE             |
+-------------------------------+
|         LOCAL NAME *          |
+-------------------------------+
|        FOREIGN NAME *         |
+-------------------------------+
```

Receive Message Block

* => Added for 1822L

Figure 5. Transmit and Receive Message Blocks

The receive and transmit blocks have each been extended hold the names (see figure 5). For as long as this connection stays open, no further translations are necessary for any messages with the same source and destination names. However, messages with different source and destination names require another connection, even if the same source and destination physical ports are used once the names are translated. This conforms with Report 4473′s requirement that different connections be established for each source and destination name pair, not just for each source and destination physical host pair. If logical addressing is not in use on the connection, the two name fields will contain the 1822L addresses of the local and foreign hosts.

In order to easily search the transmit blocks for an existing connection, all open connection blocks will sit in a hash table. As connections are opened and closed, their transmit blocks are added to and removed from the hash table. The hash table will have a number of bucket pointers typically in the range of a third of the number of transmit blocks in the IMP, to keep the potential bucket size small. A new word has been added to the transmit blocks to hold the pointer to the next block in that bucket. The hash function will only use the middle bits of the local and foreign name fields, to try to keep the results of the hash as random as possible.

When the IMP receives a regular message from a host, it first searches the transmit block hash table for a connection with the same source and destination 1822L names and/or addresses, and is from the same physical host port. If such a transmit block is found, then no further translation is needed, and the existing connection is used. If, however, no such block is found, then the destination name has to be translated and checked for effectiveness. Once this test has been passed, a GETBLK is sent to the IMP to which the translated name resolved. The GETBLK includes the source and destination logical and physical addresses. The rest of the connection setup proceeds as at present, and the messages can be sent once the connection has been established. If a host down indication is returned in reply to the GETBLK, another address from the translation database can be tried, or a dead host status can be returned to the source host if the list of possible translation has been exhausted. When the messages are ready to be sent into the destination host, the SOURCE and DESTINATION NAME fields in the 1822L leader are filled in from the information in the receive block.

Note that there is no mechanism for guaranteeing that any replies from the original destination to the source host get sent to the original source host port. If the source host wants any replies to return to the same port, the host can specify a source name that uniquely maps to that port, or it can use the 1822L address of the port in the SOURCE NAME field.

3 IMP-IMP Messages

Three IMP-IMP messages have to be changed to hold additional logical addressing information - the GETBLK, GETBLK reply, and the uncontrolled packet header. These changes are transparent to the non-logical-addressing IMPs, so that they and the logical-addressing IMPs can interoperate on the same network. In addition, there is one new IMP-IMP message, the DNA (Destination Not Available) message. This message is only used between logical-addressing IMPs.

3.1 GETBLK

As mentioned above, The GETBLK message must contain the source and destination names when a logically-addressed connection is to be opened. The source IMP simply adds the two names to the end of the existing message (see figure 6), and the destination IMP places the names into its receive block. The current GETBLK message does not use the left-hand four bits of the PKTH word, so the new GETBLK messages use left-most (most significant) bit of PKTH to signal that logical addressing is being used on this connection.

```
 15                               0
+-+-+-+-+-------+-----------------+
|0|E|D| | OCTET |   CHANNEL #     | NETH
+-+-+-+-+-+-+---+-----------------+
|0|1|C|1|T|A|   |     ACKS        | TYPH
+-+-+-+-+-+-+---+-----------------+
|       SOFTWARE CHECKSUM         | CHKH
+---------------------------------+
|           SOURCE IMP            | SRCH
+---------------+-----------------+
|   DEST HOST   |   SOURCE HOST   | SEQH
+--+------------+--------+--------+
|L*|                     |  CODE  | PKTH
+--+---------------------+--------+
|        DESTINATION IMP          | DSTH
+-------+-------+-----------------+
| HAND  | LUSE  |  LOCAL BLOCK    | MIDH
+-------+-------+-----------------+
|         SOURCE HACMEM           | DATA
+---------------------------------+
|         SOURCE HACCOM           | DATA+1
+---------------------------------+
|         SOURCE NAME *           | DATA+2
+---------------------------------+
|      DESTINATION NAME *         | DATA+3
+---------------------------------+
```

* => Added for 1822L

Figure 6. GETBLK (Code 11)

3.2 GETBLK Reply

The GETBLK reply message now needs to specify two new reasons why a connection attempt failed - the destination address wasn't authorized (in the database), or it was authorized, but it wasn't effective. The former condition can arise during database updates, when the databases in the source and destination IMPs aren't consistent. The same bits in PKTH that were unused in the GETBLK were also unused in the GETBLK reply, but will now mean the follows:

```
100000 - This is on if it was on in the GETBLK
         (signals logical addressing in use)
 40000 - The destination name was not authorized
 20000 - The destination name was not effective
```

Other than these three bits, the GETBLK reply message is identical to its current format.

3.3 Uncontrolled Packets

Since no control blocks are used when sending uncontrolled packets, they must carry all of the their overhead information along with them. Unfortunately, this includes the source and destination names, which will reside in the last two data words of the packet, just after the source host HACCOM word (see figure 7). This reduces the number of data words available for the hosts from 62 to 60. Furthermore, the existing header has absolutely no free bits that could be used to mark a packet as logically addressed. However, some flag is needed to signal a logically-addressed uncontrolled packet.

```
 15                               0
+-+-+-+-+-------+-----------------+
|0|E|D| | OCTET |   CHANNEL #     | NETH
+-+-+-+-+-+-----+-----------------+
|0|0|C|P|T|FLAGS|      ACKS       | TYPH
+-+-+-+-+-+-----+-----------------+
|       SOFTWARE CHECKSUM         | CHKH
+---------------------------------+
|           SOURCE IMP            | SRCH
+---------------+-----------------+
|   DEST HOST   |   SOURCE HOST   | SEQH
+---------------+-----------------+
|         SOURCE HACMEM           | PKTH
+---------------------------------+
|         DESTINATION IMP         | DSTH
+-------------------------+-+-+-+-+
|        MESSAGE ID       |0|0|1|1| MIDH
+-------------------------+-+-+-+-+
|            0 to 60              | DATA
|           data words            |
+---------------------------------+
|          SOURCE HACCOM          |
+---------------------------------+
|          SOURCE NAME *          |
+---------------------------------+
|       DESTINATION NAME *        |
+---------------------------------+
```

* => Added for 1822L

Figure 7. Uncontrolled Packet

The way that the presence of the logical addressing information is being signalled is based upon the fact that the SOURCE HOST field in uncontrolled packets (in the right half of SEQH) is only used to fill in the 1822 leader for the destination host. However, with logical addressing, the information is not needed for that purpose, although the contents of the field has to be preserved for a possible DNA message (see the next section). In addition, at present only 22 of the possible 256 values of source host number are presently used (16 normal hosts and 7 fake hosts), and 1822L addresses limit us to 64 hosts at most (this is a 6 bit field). So, host numbers of 200-277 octal (128-191 decimal) can be used to flag that the source host used logical addressing to send the uncontrolled packet. The low order 6 bits are used to pass along which physical port was used by the source host. Moreover, the upper 6 numbers in this range (271-277) signal that the packet was sent by a fake host with logical addressing (which may become useful in the future). Thus, the SOURCE HOST field (X in the following chart) takes on the following legal ranges (in octal):

X	Interpretation
0-77:	Packet sent by host X without logical addressing
200-270:	Packet sent by host X-200 with logical addressing
271-277:	Packet sent by fake host X+100 with logical addressing
371-377:	Packet sent by fake host X without logical addressing

Note that the ranges 0-77 and 371-377 are exactly the same as at present, and the range 200-277 corresponds exactly to the host numbers 00-77 in 1822L addresses (which have 6-bit host numbers). Also, as the number of fake hosts in the IMP grows or shrinks, the boundary line separating the real and the fake host numbers changes as well.

3.4 DNA

Logical addressing requires one new IMP-IMP message, the DNA (destination not available) message. The DNA signals that the destination of an uncontrolled packet was unable to accept the packet, either because the host was down, or the host access words didn't match, or the mapping was not authorized and effective. The DNA contains the reason why the uncontrolled packet failed, in a format similar to the GETBLK reply message (see figure 8). There is one unused end-to-end message code, type 1, code 13, which will now identify the DNA. The DNA also requires some additional information copied from the uncontrolled packet header - the source and destination names, the physical host port numbers, and the message ID.

```
 15                                0
+-+-+-+-+-+-------+-----------------+
|O|E|D|   | OCTET |    CHANNEL #    | NETH
+-+-+-+-+-+-------+-----------------+
|O|1|C|1|T|FLAGS  |      ACKS       | TYPH
+-+-+-+-+-+-------+-----------------+
|         SOFTWARE CHECKSUM         | CHKH
+-----------------------------------+
|            SOURCE IMP             | SRCH
+-----------------+-----------------+
|    DEST HOST    |   SOURCE HOST   | SEQH
+-+-+-+-+---------+---------+-------+
|L|A|E|H|                   | CODE  | PKTH
+-+-+-+-+-------------------+-------+
|          DESTINATION IMP          | DSTH
+---------------------------+-------+
|        MESSAGE ID         |       | MIDH
+---------------------------+-------+
|         DEAD HOST STATUS          | DATA
+-----------------------------------+
|            SOURCE NAME            | DATA+1
+-----------------------------------+
|         DESTINATION NAME          | DATA+2
+-----------------------------------+
```

Figure 8. DNA (Code 13)

4 NU Support for 1822L

One of the key pieces required for the 1822L implementation is the creation, storage, maintenance, and dissemination of the logical addressing translation database. This section describes how these functions will be integrated into the NU network monitoring and control system.

Of course, the most important NU function related to logical addressing is creating and maintaining the translation database. The IMP's translation database starts as entries in the network's NU database, in the host dblks, as described in section 4.1. It is then converted into NU's mobject format, which is suitable for loading into a dead IMP or for verifying against the existing databases in the IMPs, as described in section 4.2. For incremental changes to the database, a program updates the NU database and the mobject version of the translation database, and then sends an update message to each IMP in the network. This process is described in section 4.3.

An important component of the database maintenance is the mobject version of the database, which is an exact memory image of the database in the IMPs. Because the IMPs and NU start with the same database, and use the same algorithms to apply updates, the NU copy of the database will always be identical to the database in the network IMPs. Keeping this copy on NU allows the databases in the IMPs to be verified nightly (or however often as one likes), and also allows NU to detect when the IMPs' databases are filling up without code in the IMPs themselves to detect this condition. It also allows some processing to be done on NU instead of the IMPs, such as searching for free space in the ADDR table when adding a new entry.

In a typical network, the IMPs initially come up without any logical addressing translation database. The initial translation database is entered into the NU database, and the translation database is then compiled into its mobject IMP representation. After this, the database is loaded into a dead IMP in the network, which is then restarted. The other IMPs are then reloaded from this "seed" IMP, and then from each other, carrying the database along with the IMP program, until all of the IMPs in the net contain the database.

From this point on, the database can be kept up to date by sending update messages from NU to the IMPs. Whenever a change is made to the translations in the NU database, the change is reflected by update messages sent by NU to the IMPs. The IMPs update their own copies of the database. The mobject representation of the database is also updated so that it will always match the databases in the IMPs.

4.1 The NU Database and the Translation Database

The logical addressing translation database is fully integrated into the NU network database, and resides in the host dblks for the network. Each host in the network, whether singly- or multi-homed, has one dblk in the NU database. The one exception to this is if a single host is multi-homed, but wishes to be known by different ASCII names on each of its ports, in order to appear as a number of distinct hosts, then it would have a separate dblk for each of its ports. Permutations of this are possible, of course (a single machine might want to be known as BBNX on one of its ports and BBNY on two other ports, and have each be logically distinct "hosts"; the machine would have two dblks, one for each of its "hosts", and each dblk would list the ports by which that "host" was known).

Each host dblk lists the physical host ports to which that host is connected, and then lists each name, as a 16-bit unsigned number, that refers to those ports. For each name, it lists which physical port to which the name maps. The dblk also lists which of the three possible criteria should be used when performing the translation.

There is also a hashed key file, which, for each name in the translation database, contains pointers to those dblks in which that name is contained. This allows programs to quickly find all of the authorized translations for a name, without needing to search through all of the host dblks in the network database.

Once the translations have been entered into the NU database, LADBCOMP is used to compile the translation database. As it runs, LADBCOMP requests the range of IMP buffers that it should use to hold the database. This process results in an mobject representation of the database in the IMP. LADBCOMP also outputs a file containing a human-readable version of the translation database for error checking, and prints the amount of free space left for growth in the tables. If this is too small, the program can be rerun with a larger number of buffers specified to be stolen.

Since a name can appear several times in the host dblks, once for each host port to which it refers, there could be conflicts in the listed selection criterion for the name. Each listed criterion can have the value 0, 1, or 2 (see section 2.1.1 for more details). If the same name is listed as having selected more than one of the criteria, LADBCOMP selects the lowest of the listed criteria for the name.

4.2 Database Distribution

Once the database has been compiled, it has to be distributed to the IMPs. This is usually done when logical addressing is first brought up on a network, or when major changes to the database require that a new table be compiled using LADBCOMP (such as when one network is split into two, or two previously separate networks are joined into one).

To completely download the database to an IMP, the MCOPY command sends the mobject form of the database, as constructed by LADBCOMP, to a dead IMP in its loader-dumper. Once the database is in the dead IMP, the IMP is restarted and the database is distributed to the network by reloading the IMPs from each other. In this way, releasing a totally new database is similar to releasing a new version of the IMP software. Each "release" of the translation database is identified by a 16-bit serial number, which allows easy detection of an out-of-date database.

4.3 Updates and the Update Message Format

Once the database has been distributed to the IMPs, it will need periodic updates as hosts change addresses or are added to and removed from the net. An NU command, LADBUPDATE, handles the changes to the translation database. For each of the 1822L names to be updated, LADBUPDATE is given the operation to be performed (adding a new name, deleting a name, and replacing the translations for an existing name). For an addition or replacement, the input includes the complete list of new translations for that name (including host names and physical port locations). LADBUPDATE calls the standard NU database routines to change the NU database, and also updates the hashed key file to keep it current.

Once LADBUPDATE has changed the NU database, it constructs a serialized update message containing the change, and places it into a uniquely-named disk file. This file is then read to send the update to each of the IMPs in the net. The update message contains a 16-bit update serial number, which starts at one each time a new database is placed into the net. The IMPs save the number of the last update that they applied to the database. The combination of the two serial numbers allows NU to instantly spot any out-of-date IMPs. Section 5.4.1 describes the actions that the IMP takes when it receives an update message.

If an IMP had been isolated from the network for some reason during one or more database updates, it could come back up on the network with an out-of-date database. Since the database and update serial numbers are included in the status messages sent by the IMP (see section 4.4 for more information), NU will instantly detect the problem and notify the controllers. LADBUPDATE, given a suitable command-line argument, will examine the serial numbers in the IMP in question and determine which updates the IMP requires, and then send them to the IMP from the disk files in which they have been stored. If LADBUPDATE determines that serial number for the entire database is wrong, it will inform the controller that the IMP requires a reload.

An update disk file only need to be saved until all of the IMPs in the network have received and successfully processed the update. If one or more IMPs are isolated from the network, it will be saved on disk until the IMPs have been reconnected to the network and are able to receive the update.

```
 15                               0
+---------------------------------+
|             6-word              |
|              1822               |
|             leader              |
+=================================+
|             14-word             |
|            IP & UDP             |
|             leaders             |
+=================================+
|     DATABASE SERIAL NUMBER      |
+---------------------------------+
|      UPDATE SERIAL NUMBER       |
+---------------------------------+
|           1822L NAME            |
+--+-+-+-+------------------------+
|CR|G|D|N|  NUMBER OF ADDRESSES   |
+--+-+-+-+------------------------+
|    ADDR TABLE ENTRY POINTER     |
+---------------------------------+
|        1822L ADDRESS 1          |
+---------------------------------+
|        1822L ADDRESS 2          |
+---------------------------------+
|                .                |
|                .                |
|                .                |
+---------------------------------+
|        1822L ADDRESS n          |
+---------------------------------+
```

Figure 9. Update Message

Figure 9 shows the format of an update message. The update messages take the form of regular messages from the NU host to the IMP´s Logical Address Update Fake Host. The update message starts with an 1822 or 1822L 96-bit leader, followed by 14 words needed for IP and UDP headers. The UDP header contains a checksum for the entire message, and the messages use UDP port 51 (Logical Address Maintenance). The next two words contain the serial numbers for the database and the update message. This is followed by at least two words, the first of which contains the 1822L name being added, deleted, or changed. The second word contains two bits for the criterion selection, one bit that identifies this as a group address, a "delete" bit, a "new" bit, and the number of physical addresses that follow. The delete bit instructs the IMP to delete the name and its associated ADDR list from the database. The new bit identifies the name as a new insertion in the database; if the bit is off, then the name already is in the database and is being updated with a new ADDR list.

The list of physical addresses is not always necessary. For example, a name with the delete bit set would not be accompanied by a list of addresses. However, if a new list is being sent, LADBUPDATE uses the NU mobject copy of the database in the IMPs to search the ADDR table's free list to find space for the new ADDR list. If the update contains a list of physical addresses, the next word in the update has the address in the ADDR table where the physical list should be inserted. This is followed by the list of 1822L addresses that makes up the ADDR entry for the name. This list always contains all of the addresses for this name, even if the list has only one change, addition, or deletion. If this is an update of a name already in the database, then the new physical address list simply replaces the old. Otherwise, the new name and list are added to the current database. This process is further discussed in section 5.4.1.

The update fake host uses IP and UDP to reply to an update message with either an ACK or a NAK, depending on whether or not the fake host successfully applied the update. LADBUPDATE will wait for this reply after it sends an update, and will time out and retransmit the update if it doesn't receive any reply after a certain amount of time. If LADBUPDATE receives a dead IMP indication, or a NAK, or times out repeatedly when sending an update to an IMP, it will complain both to the user and to the NU log.

The ACK and NAK messages have a simple format. They will start with an 1822 leaders, followed by the 14-word IP and UDP headers, and have one word of data, which will contain a 0 for an ACK or a non-zero reason code for a NAK.

4.4 Other additions to NU

Other enhancements to NU are possible to further ease the use of logical addressing in a network, such as commands to verify the databases in the IMPs with the database stored on NU, can be implemented as time and the need arise. However, one addition that is included in the original release of logical addressing will be to include the current database and update serial numbers in both NU´s database and in the status messages sent by the IMPs every minute, with NU´s status processor checking the reported serial numbers and generating complaints on the log if they do not match. This will automatically inform the controllers that an IMP needs to be reloaded or updated, without their having to periodically check the IMPs by some other means.

5 New IMP Algorithms

This section describes the various new algorithms that are used by the logical-addressing IMP. These include both new algorithms (such as translations) and modifications to existing IMP algorithms (such as opening an end-to-end connection).

5.1 Translations

Name translations occur through one of three methods: by checking the cache (the transmit blocks), by searching the NAME table, or by using the PORT table. Each method has its own uses, described below.

The first translation routine, LGCACHK, checks the cache (the transmit blocks) for an existing connection from the same source name or address to the same destination name or address. LGCACHK takes two parameters, the source and destination 1822 names and/or addresses, and it checks the in-use transmit blocks for a match. Those transmit blocks with open connections are contained in a hash table, which can be easily searched to find a match (see section 2.2). If one is found, the transmit block number is returned. Otherwise, an error indication is returned.

Translation via the NAME table is required whenever a destination name in an outgoing (or, occasionally, a store-and-forward) message leader needs to be translated, and there isn't an open connection. Whenever such a translation is required (in host input or in TASK) the routine LGETRAN is called. LGETRAN takes as an argument the name to be translated. It returns either an error indication if the name isn't in the NAME table (the name isn't authorized), or a pointer to the first available (meaning authorized, up, and effective) ADDR entry selected according the name's criterion. It also has an error return signifying no available translations (i.e., all of the possible translations are either down or non-effective).

The calling routine then tries to use the returned address to send the message (more on this later). If this fails, then LGETRAN is again called. This returns, if possible, the next-preferential address, according to the selected criterion. If the criterion is for the closest physical address, then each time LGETRAN is called, the "distances" to the remaining effective addresses are rechecked, and the closest address is the one returned. Thus, the translations always take routing changes into account.

To perform its selections, LGETRAN uses the effective and history bits described in section 2.1.1. LGETRAN employs one of three algorithms, depending on the selected criterion:

- First reachable: LGETRAN starts its search from the beginning of the ADDR list for the name being translated. It searches in order for the first effective ADDR entry, and returns that ADDR entry to the caller. LGETRAN uses an error return if it cannot find any effective translations.

- Closest physical address: For each effective ADDR entry, LGETRAN looks up the routing distance to the host port's IMP, keeping track of which entry had the shortest distance to its destination IMP. Once this has been done for the effective entries, the winning ADDR entry is returned to the caller.

- Load leveling: This criterion uses the history bit in the ADDR entries to round-robin cycle through the ADDR entries. The ADDR entries are first checked to see if one has its history bit set. If not, then this criterion is identical to "first reachable"; LGETRAN searches for an effective translation starting from the top of the list. If one is set, it is cleared and the search starts from the following entry, wrapping around back to the top of the list if necessary. If an effective entry is found, it is returned and its history bit is set; if not (LGETRAN remembers where it started searching), then the error return is used.

In all three cases, LGETRAN interacts with the setting and clearing of the up and effective bits. Whenever either a connection attempt or an uncontrolled packet to a translated port fails because the host is down or non-effective, the corresponding bits are set in the host's ADDR entry (see sections 5.2 and 5.3). To keep the bits from locking out a down host forever, the bits are timed out and cleared every five minutes (see section 5.6).

The PORT table is used when the SOURCE NAME field in outgoing messages is checked and when NDM messages are processed. In these cases, a translation is not needed, since the IMP already knows the name and the port to which it should resolve. The information really needed is whether the name is authorized and effective and if the name matches the port that the host is using. This work is done by one routine, LGPRTCHK. LGPRTCHK is passed two parameters, the local port number and the name the host is using. It indicates whether the name matches the port in the database, and whether or not the translation is effective.

5.2 Regular Messages

When a host presents an 1822L regular message to the IMP, the SOURCE and DESTINATION NAME fields have to be translated (unless, of course, either or both is an 1822L address). First, LGPRTCHK is called to check the validity of the SOURCE NAME field. If this test is passed, then LGCACHK is called to find an already-open connection for the message. If this fails, then LGETRAN is called to translate the destination name. Assuming that LGETRAN returned a valid translation, a GETBLK is sent to the destination IMP. If the GETBLK was successful, then the rest of the current end-to-end procedure is followed, and the message is sent. If the GETBLK failed, the down and non-effective bits in the ADDR entry returned by LGETRAN are set as necessary, LGETRAN is called to return another translation, and another GETBLK is attempted. This procedure continues until either a connection is opened, in which case the transmit block is added to the hash table, or until there are no other effective translations. Once the host code gets through the translation procedure, the rest of the end-to-end protocol continues as at present.

In the destination IMP, things proceed pretty much as they do now. When the GETBLK is received, the usual HAC and host up-down checks are made. If these checks pass, the destination name is then checked in the destination IMP's PORT table for authorization and effectiveness. If this check also passes, the IMP tries to get a receive block. If successful, the IMP copies the information of interest, including the 1822L names, from the GETBLK, and sends a successful GETBLK Reply to the source host. However, if any of these steps fails, then an unsuccessful GETBLK Reply is returned. Included in the reply is why the GETBLK failed.

5.3 Uncontrolled Packets

Up to now, a source host sending uncontrolled packets has not received any indication of whether the destination host was dead or alive. New code is being added to let the source host know if the destination host is dead or non-effective. When the IMP receives an uncontrolled packet from a host, LGPRTCHK is called to check the source name. Next, LGETRAN is called to find an effective translation for the destination name. If this succeeds, then the packet is sent. If this fails, the host will receive either a type 15, subtype 3 message, a type 7, subtype 1 message, or a type 15, subtype 5 message, depending on why LGETRAN failed.

The host code in the destination IMP will now return a DNA message (see section 3.4) if the destination host is down or non-effective. If the source IMP receives a DNA, a type 15, subtype 6 message is sent back into the source host, and the translated address is marked dead and/or not effective, as appropriate. There is no attempt to re-send the packet to another host in the list.

Finally, TASK has been extended to handle the case where an uncontrolled packet is received by a tandem IMP, which discovers that the originally translated address' IMP is not reachable. In this case, TASK will retranslate the name and try to find a reachable, up, and effective destination. If it finds one, the packet is sent on its new way. If not, the packet is discarded, without any indication returned to the source IMP.

5.4 Translation Database Updates

Translation database updates have been already discussed in some detail. There are two types of updates: a full "release" of a new database, as discussed in section 4.2, and updates employing update messages, as was described in section 4.3. Section 5.4 further details the process that occurs in the IMP processing such an update.

Whenever an update occurs, the database has to be locked so that translations cannot occur while the update is processed. This is further discussed in section 5.4.2.

When an IMP is restarted or reloaded, or receives an update that involves a host port at the local IMP, the PORT table has to be rebuilt. This is discussed in section 5.4.3.

5.4.1 Updates

When an update message is received, the IMP´s IP and UDP support modules are called, which start to process the message by building the corresponding parts of the reply to be sent in response to the message. Next, the update message´s serial numbers are checked against the database´s current numbers, and if the update is found to be out of sequence, a NAK is sent back to NU, and the update is discarded (if the serial numbers are identical to those in the last successfully processed update message, then this message is assumed to be a duplicate, and an ACK is sent). Otherwise, the database is locked, as discussed in the next section, and the update is processed.

The update is handled in the obvious way, as dictated by the database format as described in section 2.1.1. First, the update is checked to see if it includes a list of physical addresses (deletion requests will not include such a list). If the update does, the address where NU placed the list in the ADDR table is checked to make sure that the space is free and large enough. If the checks succeed, the space is allocated and the list of physical addresses is copied into it. If the space is too small or unavailable, a NAK is generated and the update is discarded.

Following this are three main cases:

1. The name in the update is not yet in the database. In this case, the spot in the NAME table where the name should be inserted into the database (in sorted order) is found; the rest of the NAME table is moved down two words to make room

for the new entry; the information is copied into the table; and the new entry is pointed to its associated ADDR list, which has already been placed into the ADDR table.

2. The name in the update is in the database, and this is not a deletion request. If there was a new ADDR list in the update, the old list is freed and the NAME entry is pointed to the new list. The criterion and group bits in the NAME entry are copied from the update.

3. This is a deletion request. The name is found in the NAME table, its associated ADDR list is freed, and the following portion of the NAME table is moved up by two words to fill in over the deleted entry.

If a name is being deleted, or an ADDR list is being changed so that a reference to a local host port is being deleted, then the PORT table has to be updated (see section 5.4.3), and any open connections that use that translation have to be closed.

If the update is successfully processed, then the fake host sends an ACK message back to NU. However, if an update fails and is discarded for any reason, then the IMP send backs a NAK. Also, the IMP clearly doesn't have the complete set of updates from that point on. The fact that the IMP is now out of date is quite visible on the NU log from both the effects of the NAK and from the database and update serial numbers in the status messages from that IMP.

5.4.2 Database Locking

As already mentioned, the database needs to be locked during updates so that translations are not attempted while the database is in flux. There are three main ways to lock the database: by inhibiting all interrupts, by inhibiting the host code from running, and by using a software semaphore to disallow translations.

Inhibiting all interrupts while processing update messages is the easiest lock to implement, but problems could occur if an update requires a large number of NAME table entries to be moved to insert or delete a NAME entry. For example, a long inhibit strip could easily have an adverse affect on the IMP´s modem-in code, so this is not a very practical solution.

Another solution would be to add a mechanism to the IMP to keep the host processes from running, while allowing the rest of the IMP (or at least the modem code) to function. The database update code could run at a higher priority than the host code, but at a lower priority than the modem code. However, this could have some interaction with the timeout-driven functions in the

IMP, and also penalizes the non-logical-addressing hosts unfairly.

The best solution is a software semaphore that, while set, signals the host code to not perform any translations. This semaphore is checked in LGETRAN whenever a translation is to be performed, and causes the host process to wait for the semaphore to clear before proceeding with the translation.

Whenever an update message is being processed, the semaphore is set. This causes the host code to debreak for a certain amount of time. When the host process returns from the debreak, it recalls LGETRAN to check the semaphore and waits again, if necessary, until the semaphore is clear and the translation performed. This effectively blocks the host during the update. However, an update really shouldn't take that long to process; the amount of time to perform the update is dominated by the time it takes to insert a name into or a delete a name from the NAME table. Assuming a worst-case table size of 2000 names, then the average insertion or deletion will require 1000 names (2000 words) to be moved. The NMFS BLT (BLock Transfer) instruction requires 15+7n 125ns cycles to run, where n is the number of words being moved. In this case, the transfer would need (15+7*2000)*.125 us, or 1752 us (1.752 ms). A worst-case insertion or deletion would require twice that, or 3.5 ms. This is a negligible wait compared to some of the other waits that the host code uses when gathering resources.

Since the host code runs at a higher priority than the fake host performing the update, and since there will be no debreaks in the middle of a translation, there are no race conditions to worry about. The semaphore has the additional advantage of locking out only those hosts that attempt to use logical addressing while an update is taking place. Those hosts that don't use logical addressing, or that don't send or receive any logically-addressed messages during the update, will be totally unaffected.

5.4.3 Building the PORT Table

Every time the IMP is restarted or reloaded, the PORT table (which is a local entity to each IMP) has to be completely rebuilt by the IMP's initialization code. One pass will be made through the entire database, during which all twenty-three PORT lists will be constructed. Section 2.1.2 contains descriptions of the PORT table and the PORT lists.

The PORT table will be initialized to have all twenty-three of its pointers set to point to lists that each consist of one dummy entry. Once the PORT table pointers have been set, The ADDR list for each NAME entry is checked to see if any of the addresses for the name refer to host ports at this IMP. If a match is found, then the name is added to the end of the appropriate PORT list (if that port´s list only had a dummy entry, then the dummy entry is re-used to become a real entry). This process continues until the entire database has been scanned.

Whenever an individual update involving a new list of addresses occurs, the new and possibly old lists are checked for local hosts. If a local host reference is being deleted, then it is removed from the correct PORT list (if it was the only entry in the list, then it becomes a dummy entry, otherwise it is actually removed). If a host reference is being added, then it is placed in correct PORT list. No other change to the PORT table needs to be made as a result of processing update messages.

5.5 Additional Host Code Processing

There are a number of places where new functionality has been added to the host input and output processes. In most, if not all cases, however, this new functionality should not require a large amount of new code. For example, there already exists an easy-to-use mechanism in the host code for host input to send replies to a host based upon a message received from the host, by placing the reply in the message´s transaction block, and then placing the block on the host reply queue for the host. The use of this mechanism greatly simplifies the implementation of this new functionality, since it mostly involves new messages types between the IMP and the hosts.

One example of this new functionality is in the processing of the three new host-to-IMP message types, Name Declaration Messages, Name Server Requests, and Port List Requests. All three message types result in a message being sent back to the host, once the message itself has been processed by host input. NDMs, for instance, cause effective bits in the PORT table to be set or reset, and all three messages result in replies being sent back to the host. In each case, the body of the reply message is constructed by host input, a transaction block specifying the type of reply and a pointer to its body is placed on the host reply queue, and host output, when processing the reply queue, sends the required leader and the reply to the host. In the existing IMP, all such host replies are just leaders, and are sent directly from leader area, but it shouldn't be hard, by using chained IOCBs, to also send the body of the replies for each of these messages once the leader has been sent.

Another example of new functionality is in the expanded handling of raw messages. Again, the transaction blocks can be used to send a reply back to the host when it is now necessary, without much new code needing to be added.

5.6 Timing out the Effective and Up/Down Bits

A new process will be added to the IMP that will take care of timing out the effective and up/down bits in the ADDR entries. Every five minutes, each ADDR entry should have its effective and up/down bits cleared so that messages can again be attempted to be sent. This will be a very low-level process, running right above background level.

When the process wakes up at the beginning of a run through the ADDR table, it clears the bits in 100 entries and then sleeps for 1 second. When it next wakes up, it does the next 100 entries, and then sleeps for another second. With a maximum of 8192 entries in the table, this process should take a maximum of 2 minutes or so to perform, even with a large number of interruptions from the higher levels in the IMP. The process then sleeps for the remainder of the 5 minutes.

Based upon operational experiences, all of the timeout parameters discussed above will be easily tunable to improve the performance of logically addressed messages in the net.