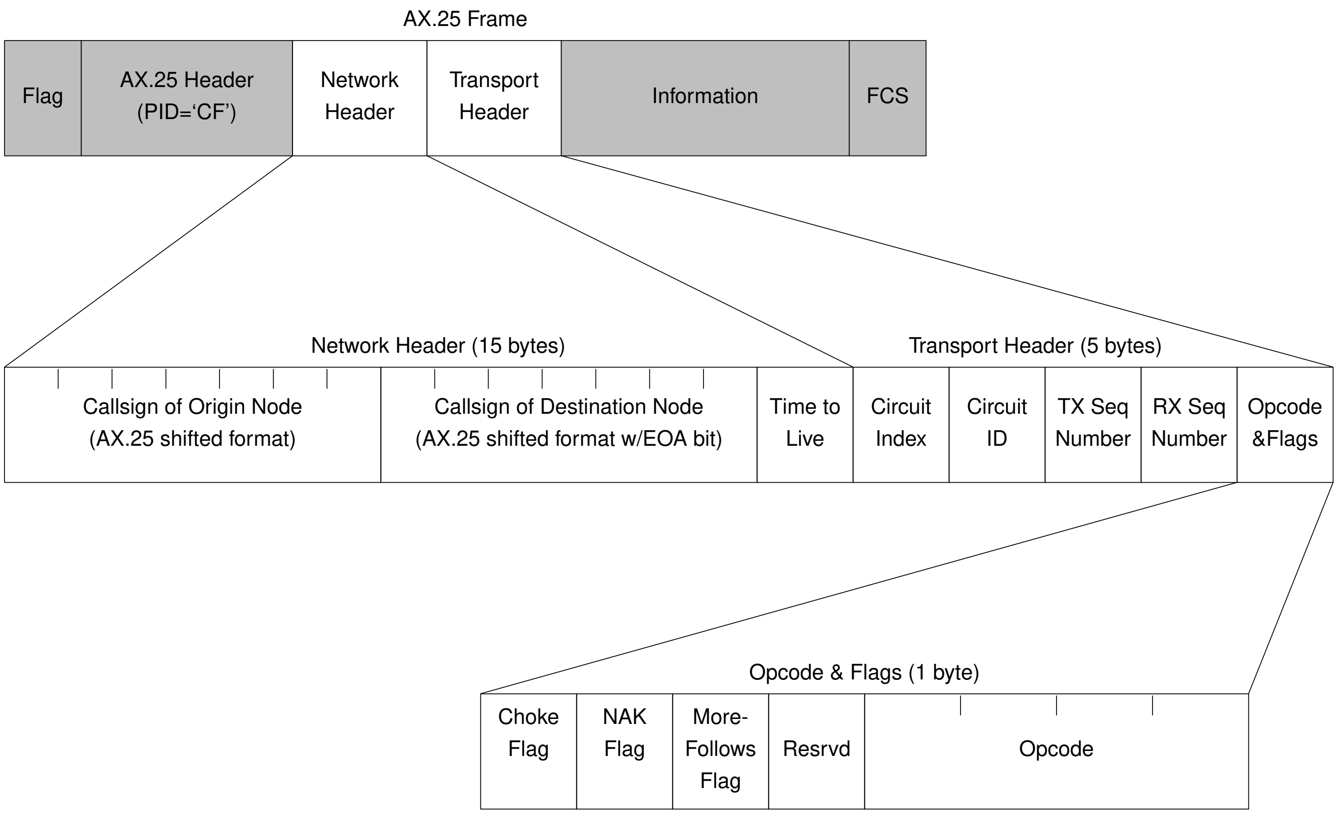
The NET/ROM Protocol

This chapter describes the higher-level protocols employed by NET/ROM. It assumes a knowledge of the underlying AX.25v2 link-layer protocol. [A detailed description of AX.25v2 is available from the American Radio Relay League, Newington, CT 06111, USA.]

Structure of Inter-Node HDLC Frames

The HDLC frame structure used by NET/ROM for its inter-node crosslinks is illustrated in the following diagram:



Each frame consists of a standard AX.25 link header, followed by a 15-byte network header and a 5-byte transport header. The network header contains the information needed for automatic routing, while the transport header supports end-to-end error and flow control for circuits.

In amateur radio service, NET/ROM identifies its inter-node frames with an AX.25 protocol identifier byte (PID) of ‘CF’ hex. In commercial service, various other PID values are used.

Transport Layer (End-to-End) Protocol

The Circuit Manager implements a conventional “sliding window protocol” in order to provide end-to-end flow and error control for each transport circuit. The protocol is similar to AX.25 but with these major differences:

1. The receive window size is negotiated, and is usually greater than 1.
2. Message sequence numbers are 8-bit fields, allowing window sizes up to 127 frames.
3. Selective NAKing is supported.

Six transport-layer message types are supported:

Connect Request
Connect Acknowledge
Disconnect Request
Disconnect Acknowledge
Information
Information Acknowledge

These are described in greater detail below.

Connect request

My Circuit Index	My Circuit ID	(Unused)	(Unused)	Opcode ='x1'	Propose Window Size	Callsign of Originating User (AX.25 shifted format)	Callsign of Originating Node (AX.25 shifted format)

A Connect Request is used to initiate a new transport circuit between the originating node and the destination node, on behalf of the originating user.

The *circuit index* is the subscript of a circuit table entry in the originating node. The *circuit ID* is a serial number used to qualify the circuit index, in order to eliminate any possible ambiguity in identifying the circuit. The *proposed window size* specifies the maximum receive window size (in frames) that the originating node is prepared to accomodate.

Connect acknowledge

Your Circuit Index	Your Circuit ID	My Circuit Index	My Circuit ID	Opcode ='x2'	Accept Window Size

A Connect Acknowledge is used to respond to an incoming Connect Request. If the high-order bit of the opcode byte is set, it indicates that the Connect Request is being refused; otherwise, it is being accepted. The *accepted window size* indicates the negotiated size of this circuit, and will never exceed the *proposed window size* of the Connect Request.

Disconnect request

Your Circuit Index	Your Circuit ID	(Unused)	(Unused)	Opcode ='x3'

A Disconnect Request is used to request the termination of a transport circuit, and may be sent by the node at either end of the circuit.

Disconnect acknowledge

Your Circuit Index	Your Circuit ID	(Unused)	(Unused)	Opcode ='x4'

A Disconnect Acknowledge is used to acknowledge a Disconnect Request.

Information

Your Circuit Index	Your Circuit ID	TX Sequence Number	RX Sequence Number	Opcode ='x5'	Information

An Information message is used to pass user information across a transport circuit.

Because AX.25 frames are limited to 256 bytes and the combined network and transport header overhead totals 20 bytes, the maximum size of the information field is 236 bytes. NET/ROM automatically fragments and reassembles any user supplied link-layer information frames that exceed 236 bytes in order to meet this constraint.

Each Information message also serves as a "piggybacked" Information Acknowledge. The *Tx Sequence Number* identifies the current information, and the *Rx Sequence Number* specifies the next incoming information expected.

If the *choke flag* is set (bit 7 of the opcode byte), it indicates that this node cannot accept any more Information messages until further notice. If the *NAK flag* is set (bit 6 of the opcode byte), it indicates that a selective retransmission of the frame identified by the *Rx Sequence Number* is being requested. If the *more-follows flag* is set (bit 5 of the opcode byte), it indicates that the information is a fragment of a long information frame, and must be reassembled with one or more following information messages by the destination node.

Information acknowledge

Your Circuit Index	Your Circuit ID	(Unused)	RX Sequence Number	Opcode ='x6'

An Information Acknowledge is used to acknowledge incoming Information messages. The *Rx Sequence Number* specifies the next incoming information expected.

If the *choke flag* is set (bit 7 of the opcode byte), it indicates that this node cannot accept any more information messages until further notice. If the *NAK flag* is set (bit 6 of the opcode byte), it indicates that a selective retransmission of the frame identified by the *Rx Sequence Number* is being requested.

RS232 Interconnect Protocol

For multi-channel nodes, information is passed among the interconnected TNCs via their RS232 ports. The interconnect operates using the same link-, network-, and transport-layer protocols as HDLC crosslinks, except that a simple asynchronous variant of HDLC is employed.

Each frame is preceded with an ASCII STX (instead of an HDLC flag), and terminated by an ETX plus a one-byte checksum (instead of an HDLC frame-check sequence). Any embedded STX, ETX or DLE characters within the body of the frame are prefixed by a DLE character (this takes the place of the normal HDLC "bit stuffing").

When two TNCs are connected using the recommended TNC-to-TNC cable, communications between the two TNCs is full-duplex and extremely fast (especially at 9600 baud).

When three or more TNCs are interconnected using the recommended diode-matrix coupler circuit, intercommunications between the TNCs is essentially half-duplex and utilizes CSMA/CD arbitration. Consequently, do not expect performance to be quite as spectacular as it is with the dual-channel configuration.

Routing

This chapter provides a detailed description of the internal workings of NET/ROM's automatic routing mechanism.

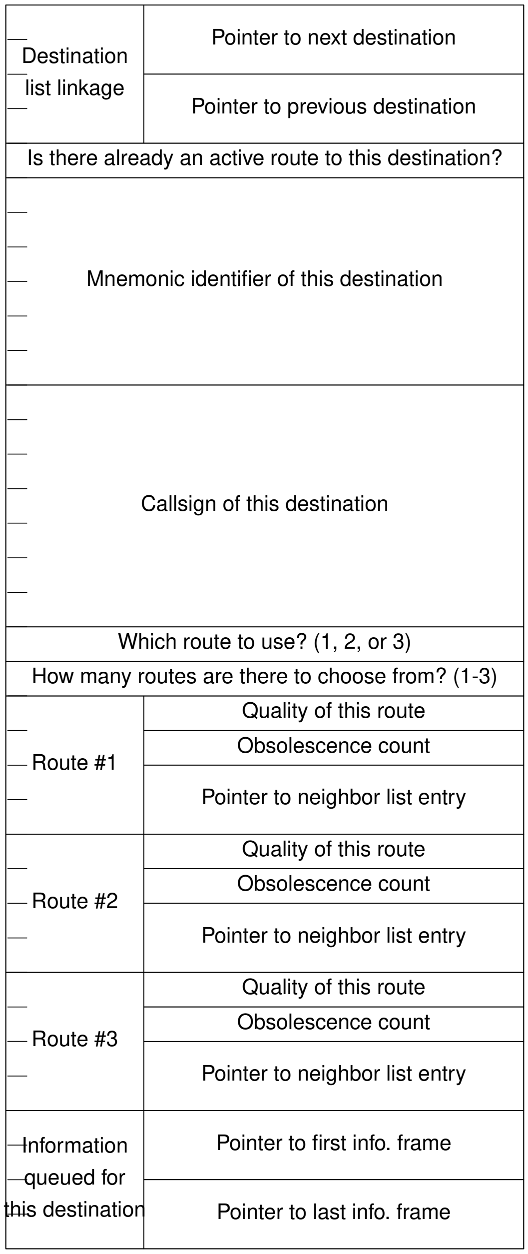
Routing Table Structure

The routing table maintained by each node consists of two dynamically allocated threaded lists: the *destination list* and the *neighbor list.* The destination list contains an entry for every other node "known" to this node - this is the list displayed by the NODES command. The neighbor list contains an entry for only "neighboring" nodes to which this node has a *direct* crosslink - this is the list displayed by the ROUTES command.

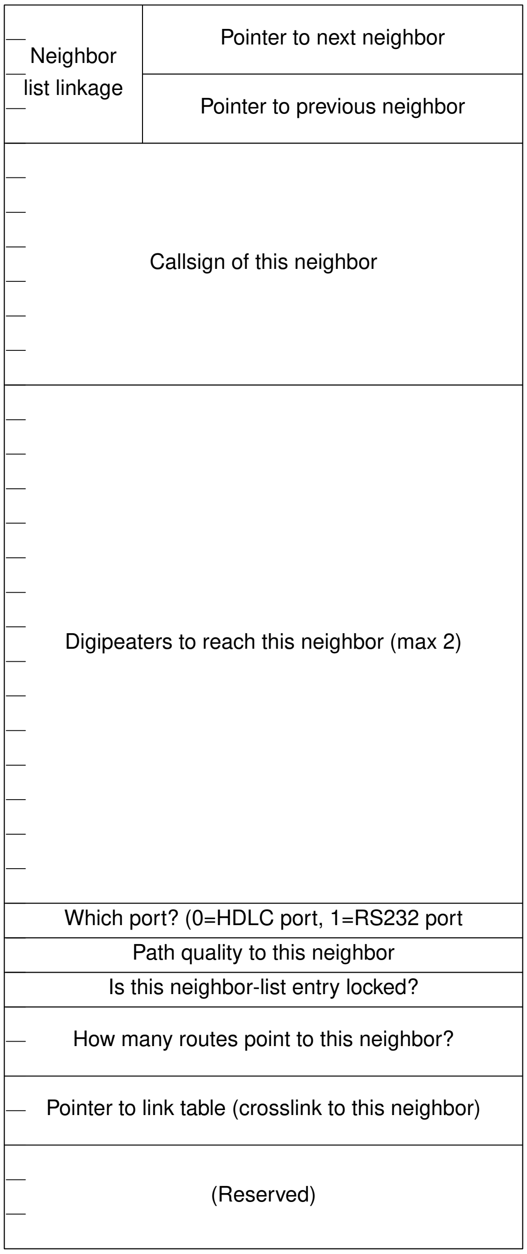
For each node in the destination list, the routing table up to three *routes* to that destination node. In this context, a route simply identifies a neighboring node that is a step closer to the ultimate destination. For each route, the destination list maintains a *quality* value which quantifies the relative desirability of each route. NET/ROM maintains route quality as an integer which ranges from 255 (best) to 0 (worst). Routes are maintained in sorted order by quality, and NET/ROM always attempts to use the highest-quality route available. It also keeps an *obsolescence count,* which enables NET/ROM to purge paths from its routing table when it has become unuseable and remained so for a protracted period of time.

Observe that the routing table does *not* contain the entire path to each known destination (this is unnecessary, and would require too much memory), but just a list of neighboring nodes that are reasonable choices for a next step enroute to the destination. You can ask to see the various routes to a particular destination node by doing a NODES command that specifies the callsign or mnemonic identifier of the destination.

Destination List

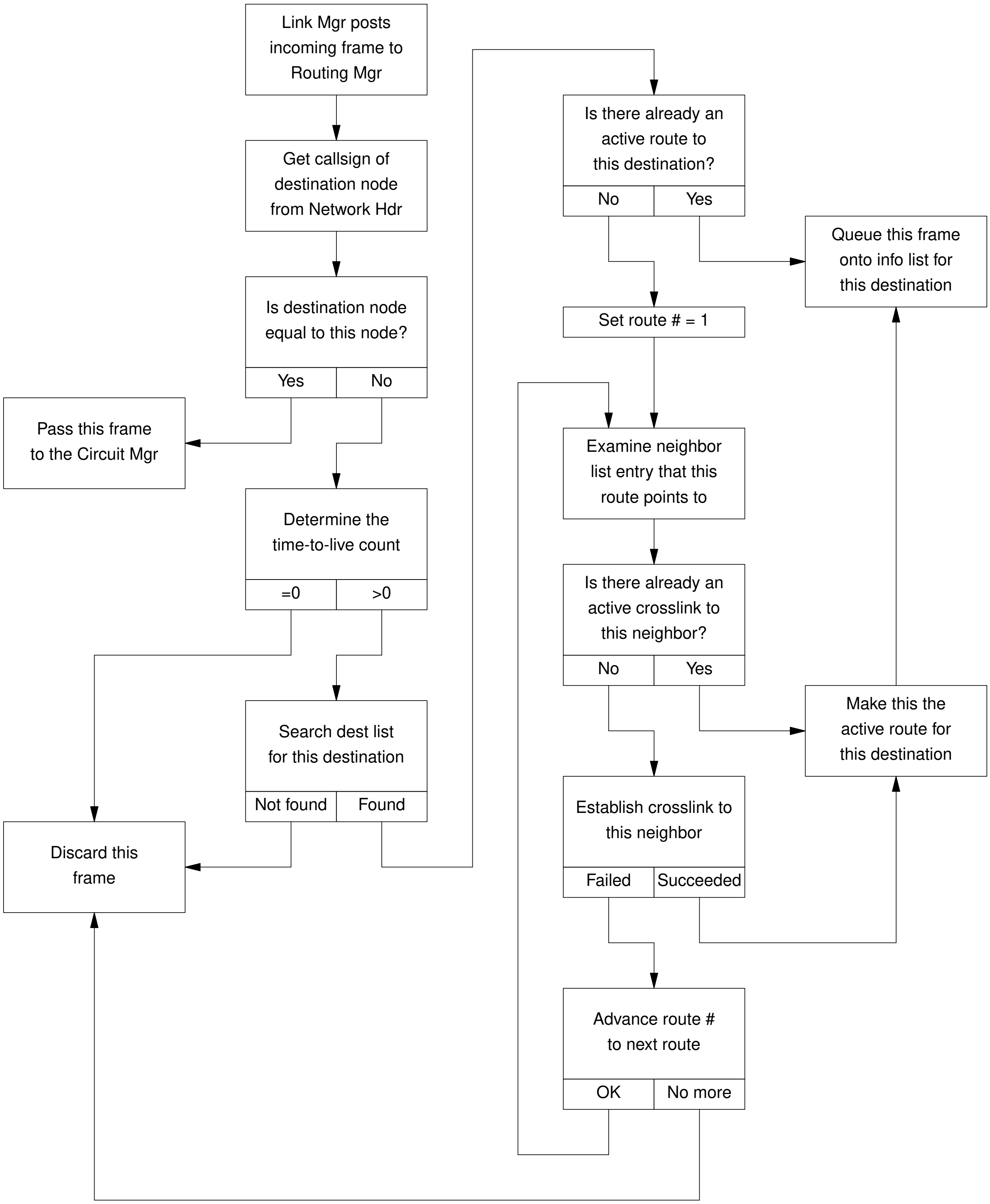


Neighbor List



Routing Algorithm

The Routing Manager analyzes the network header of each incoming crosslink-frame, and determines how to route that frame. The routing algorithm is straightforward, and is summarized in the following (somewhat simplified) flow diagram:


Link Mgr posts incoming frame to Routing Mgr
Get callsign of destination node from Network Hdr
Is destination node equal to this node?
Yes
No
Pass this frame to the Circuit Mgr
Determine the time-to-live count
=0
>0
Search dest list for this destination
Not found
Found
Discard this frame
Is there already an active route to this destination?
No
Yes
Queue this frame onto info list for this destination
Set route # = 1
Examine neighbor list entry that this route points to
Is there already an active crosslink to this neighbor?
No
Yes
Make this the active route for this destination
Establish crosslink to this neighbor
Failed
Succeeded
Advance route # to next route
OK
No more

Automatic Routing Table Updates

Each node makes a periodic broadcast of information from its routing table in order to provide the basis of routing table updates at neighboring nodes. The broadcast is normally made once an hour, although the frequency may be changed by the control operator.

The routing broadcast takes the form of one or more AX.25 UI-frames tagged with a special PID ('CF' hex in the amateur radio service). The source callsign identifies the broadcasting node, and the destination callsign is "NODES". The information contents of each frame has the following structure:

Signa-ture 'FF' hex	Mnemonic ident of sending node (6 bytes)	Callsign of destaination node (7 bytes)	Mnemonic ident of destination node (6 bytes)	Callsign of best-quality neighbor (7 bytes)	Best-quality value	(more destinations)

Repeat for each known destination
(up to 11 per UI frame)

Up to eleven destinations are packed into each UI-frame, and the node broadcasts as many such frames as required to send its entire routing table.

When a node receives an auto-routing broadcast UI-frame from one of its neighboring nodes, it analyzes the contents and makes appropriate updates to its own routing table. This process is more complex than one might think. The receiving node utilizes a series of heuristic rules to keep the size of the routing table manageable and to try and avoid "loops" and other undesirable routes. Here is a summary of the most important rules used in processing auto-routing update broadcasts:

1. If the worst quality for auto-updates parameter is zero, all auto-update broadcasts are ignored.
2. If the UI-frame does not have the proper PID or signature byte, the frame is ignored.
3. The neighbor list is searched for an entry corresponding to the node that originated the broadcast. If no such entry exists, a new one is created and assigned a default path quality in accordance with the quality parameter appropriate to the channel (HDLC or RS232) over which the broadcast was received.
4. A direct route is assumed to exist to the node that originated the broadcast. The quality of this route is taken from the path quality in the neighbor-list entry.
5. For each destination node listed in the broadcast, an indirect route is assumed to exist via the node who originated the broadcast. The quality of this route is calculated by combining the route quality from the broadcast with path quality of the neighbor node as defined in the neighbor-list entry. The qualities are multiplied, normalized, and rounded as shown in the following formula:

   routequality = [(broadcastquality x pathquality) + 128] / 256

6. An indirect route is considered to be a trivial loop if the callsign of the best-route neighbor node in the broadcast matches the callsign of the receiving node. Trivial loop routes are assigned a quality of zero, so they are used only as a last resort. Quality-zero routes are never included in outgoing auto-routing broadcasts.

7. Only the three highest-quality routes to a destination are retained.
8. Any route with quality less than the *worst quality for auto-updates* parameter is ignored.
9, If the number of entries in the destination list is greater than or equal to the *max destinations* parameter, then no additional destinations will be added.

Each route in the routing table has an *obsolescence count* which is initialized to the *obsolescence count initializer* parameter whenever the route is added or updated as the result of an auto-routing broadcast. The count is also reinitialized to this value whenever the route is actually used successfully. The default initializer value is 6. Periodically, NET/ROM scans through the routing table and decrements the obsolescence count of every route in the table - this scan occurs with same frequency as routing broadcasts, normally once each hour. If the obsolescence count of a route is decremented to zero, the route is deleted from the routing table.

A routing table entry created by the automatic routing update mechanism can never have an obsolescence count of zero, since such an entry is automatically purged from the table when its count reaches zero. When a route is entered into the routing table manually with the NODES+ command, however, it is possible to set the route's obsolescence count to zero. This has special significance: it marks the route as *locked.* such a locked route will never be updated or deleted by the auto-update mechanism. It can, however, be deleted manually with the NODES- command.