TRANSMISSION OF IP DATAGRAMS OVER NET/ROM NETWORKS

Daniel M. Frank, W9NK
1802 Keyes Avenue
Madison, WI 53711-2006

ABSTRACT

One of the main design goals of the Internet Protocol was that IP datagrams could be carried over existing local- and wide-area networks. This characteristic of IP makes it possible to build so called "internetworks" out of existing network facilities. We built support for an existing Amateur wide area network, NET/ROM, into the KA9Q TCP/IP package, allowing the use of NET/ROM to carry IP datagrams, and adding features which make the KA9Q software useful as a full duplex NET/ROM packet switch. We have also shown that NET/ROM may be used as a datagram network only, independent of its transport and application layer facilities.

INTRODUCTION

In the late Seventies and early Eighties, the world of computer communications consisted of many isolated local- and wide-area networks. Enough communications capacity existed to link the entire country, and much of the world, into a single large network, but the existing facilities were physically and logically dissimilar. They could not simply be "plugged together" to make this large "internetwork" (or "Internet").

The designers of the Internet Protocol (the "IP" in "TCP/IP") were committed to overcoming the obstacles that prevented an Internet from developing. They came up with two key ideas:

o **Gateways** can be established between networks. A gateway is a computer which possesses the physical resources and software to connect with and speak to more than one kind of network.

o A single protocol can be developed whose messages ("datagrams") can pass through any network, hidden inside that network's "native" messages. When a message with a datagram inside it encounters a gateway, the gateway "unwraps" the datagram, rewraps it in the native message of a second network, and sends it on its way. If a datagram is too large to fit inside the native message type of a network, the gateway breaks it into pieces ("fragments"), each of which is then wrapped in a native message and sent on.

By using IP and gateways, the designers of the Internet have created a global "network of networks", which today encompasses hundreds of thousands of computer systems, connected to every conceivable kind of network.

Amateur packet radio networking, like computer networking, consists of many different network technologies and protocols. Local AX.25 communications, digipeating, TexNet, ROSE, and NET/ROM, to name only a few, coexist or compete for dominance as the network technology of choice, This competition is healthy, and is in the spirit of amateur radio experimentation. Any attempt to establish one network over another as the single "standard" is both pointless and doomed to failure. No single standard can ever be imposed on radio amateurs any more than it could have been imposed on computer networks, given the investments already made in equipment, software, and education.

As the dissimilar amateur networks grow in size, they meet up with each other. Sometimes they coexist on the same channels. But without gateways and some kind of Internet Protocol, each network is an island of communication, unable to send or receive data beyond its own shores.

The work described in this paper is a first step towards true Amateur Radio internetworking. Using the KA9Q TCP/IP package as a basis, we have built a software system which functions as a gateway between local TCP/IP networks and the NET/ROM network. It allows IP datagrams to be forwarded automatically and transparently across existing NET/ROM facilities. In addition, as a full implementation of NET/ROM layer 3, it is capable of functioning as a NET/ROM relay node (as opposed to an AX.25 endpoint), and as a full duplex NET/ROM packet switch.

IP OPERATIONS OVER STANDARD AX.25 CONNECTIONS

In order to properly understand how we have interfaced to the NET/ROM network, we should first examine how "ordinary" TCP/IP operations take place over AX.25. This description follows the IS0 OSI Reference Model (RM), a seven-layer classification of network facilities.

From the bottom up, the layers used in packet TCP/IP operation are:

(1) A **physical layer,** made up of the radios, antennas, and modems used to generate and carry the tones used to convey digital data from one place to another.

(2) A **data link layer,** made up of HDLC and AX.25, used to format and address the data, detect errors and discard bad packets. The link layer only knows about and communicates with stations with which we are directly connected. In the case of packet radio, this means stations with which we have reliable, direct communications. (Digipeating doesn't count, for purposes of this

discussion.)

(3) A network **layer,** responsible for routing packets to their destinations through one or more link-to-link hops. The main distinction between the data link and network layers is that the network layer provides facilities for communication between stations not directly connected. The network layer has to have some concept of **routing,** that is, the path to be taken by a packet to reach its destination. We use IP as our network protocol.

(4) A **transport layer,** responsible for reliable end-to-end communications. Our network layer does not guarantee that a packet will actually reach its destination. While AX.25 provides link layer acknowledgement and retransmission, it does not guard against nodes which go down, software errors, or a destination station which is not on the air. The transport layer provides for an acknowledgment to be sent from the packet's ultimate destination, and for retries in case that acknowledgment doesn't arrive within a reasonable amount of time.

The other function of a transport layer is **multiplexing.** The network layer provides only host-to-host addressing. However, a computer can have many users, and provide many different services. The transport layer takes incoming packets from the network and directs them to the proper programs based on information contained in the **transport header** portion of the packets.

Our transport protocol is TCP, the Transmission Control Protocol.

(5) **The session layer** is mainly involved with providing services to individual programs within the computer. It is not of importance for the current discussion.

(6) The presentation layer is mainly concerned with the uniform formatting of data, or its conversion between different character sets. Some of the TCP/IP user programs have a very simple presentation **"layer"** which maps plain text messages in the native character set of the **user's** computer, to and from ASCII with a standard **line-**ending convention.

(7) The application layer is made up of the various programs and services that use networking facilities. Users of TCP/IP mainly make use of **telnet,** for keyboard to keyboard chat and remote **login, smtp** for automated transfer of mail, and **ftp,** for easy exchange of files.

LINK LAYER MULTIPLEXING

As can be seen from our description, local TCP/IP operation uses regular AX.25 communications for its link layer. An AX.25 packet containing an IP **datagram** contains a special code in the protocol ID (PID) field of its header. This allows the link layer software to forward the contents of the packet to the proper part of the KA9Q package, in this case the IP routing code.

If the AX.25 packet contained a PID of **"no** level **3",** the link layer would forward it to a different part of the package, in this case the AX.25 session code, which allows users of the package to hold **"regular"** AX.25 conversations, bypassing all layers between the link and application layers. (This brings up an important point about the reference model **we've** presented: an implementation may not contain certain layers from the RM if the services they would have provided are unused or unneeded.)

This switching of packets at the link layer based on their **PIDs** is known as **link layer multiplexing.** Multiplexing at the link layer is extremely useful, because it allows different network layer protocols to share the same data link services, and often the same link connections. Link layer multiplexing is what allows the KA9Q software with NET/ROM support to act as a digipeater, an IP relay, and a NET/ROM relay node, all on the same channel, through the same TNC.

AN OVERVIEW OF NET/ROM

Now that we understand how IP datagrams are carried over packet radio links, we should examine how NET/ROM operates. Again, we will use the IS0 OSI Reference Model as our framework:

(1) The **physical layer** is the same, i.e. radios, antennas, and modems.

(2) The data link layer is again **AX.25,** but the Protocol ID field of NET/ROM packets is set to a special NET/ROM ID.

(3) The network layer of NET/ROM handles the automatic routing of packets to their destination. A NET/ROM network packet header contains the source and destination callsigns of the NET/ROM endpoints. There is no information about the route the packet will travel to its destination. Instead, every node maintains a routing table based on routing **adjacencies:** it receives broadcasts from other nodes which **say,** essentially, **"I** am willing to take traffic for such-and-such a **node."** When a NET/ROM node receives a network packet, it examines its routing table to see if anyone is willing to pass it on toward its destination. If so, it hands off the packet to the next station. If not, it simply throws the packet away without comment.

The type of network communications service (as opposed to the routing techniques) used in NET/ROM (and IP) is usually called an **unreliable, connectionless datagram layer,** and the network layer packets are generally called **datagrams. The** service is **unreliable,** because it does not guarantee or confirm ultimate delivery. It is **connectionless,** because no **circuit** is established over which datagrams will travel. (This contrasts with some public data network protocols, where before data may be sent to a remote system, a fixed path to that system must be set up through the network, with resources **preallocated** at every intervening node. Each approach has its advantages and adherents.)

(4) The transport layer of NET/ROM uses what is called a sequenced packet protocol. Unlike TCP, which delivers an unsegmented stream of bytes to the receiver, and is free to pack as many or as few bytes into each message as it likes, the NET/ROM transport delivers a sequence of packets. The amount of data in these packets is determined by the amount of data in the AX.25 packets the NET/ROM user presents for transmission. NET/ROM is not free to combine packets together for greater efficiency, although it can fragment and reassemble packets' which are too large to fit in one of its transport messages.

The NET/ROM transport protocol provides end-to-end delivery and acknowledgement, as well as demultiplexing of arriving messages by circuit number. A NET/ROM node can be handling traffic for more than one circuit, or connection, at a time, and it directs that traffic internally by examining the circuit number field of the transport header.

(5) The session layer is not present in NET/ROM.

(6) The presentation layer is not present in NET/ROM.

(7) The application layer is what a user sees when he or she connects to a NET/ROM node. It is responsible for responding to user commands to list routes and nodes, and establish connections. "No layer 3" AX.25 packets arriving at a NET/ROM node are shunted directly up to the application layer, while "NET/ROM" PID packets are forwarded up to the NET/ROM network layer. This link layer multiplexing should be familiar from our earlier discussion.

A full explanation of how the NET/ROM software works is beyond the scope of this paper. The reader is referred to the NET/ROM manual for further details.

NETWORK AND INTERNETWORK

Our presentation of the IS0 OSI RM has been somewhat simplified. In particular, the IS0 recognizes a subdivision of Layer 3 into a Network Layer (3A) and an Internetwork Layer (3B). Strictly speaking, the Internet Protocol (IP) is a 3B protocol, while the NET/ROM network service is a 3A protocol. To put it somewhat crudely, IP is an Internetwork Layer because its messages can be routed through multiple logically and physically distinct networks. The same cannot be said of X.25, for example, or of NET/ROM layer 3. Our NET/ROM support in the KA9Q package reflects this distinction.

The KA9Q NET/ROM software is not a full NET/ROM implementation. That was unnecessary for our purposes. We didn't need the NET/ROM transport protocol, since our reliable end-to-end services are already provided by TCP. We didn't need the application layer, for similar reasons. What we did need was an existing network service that could carry our IP datagram traffic to remote destinations simply and easily. The NET/ROM network layer was sufficient for this purpose.

We use the NET/ROM nodes as a datagram network. When we have traffic to pass through a local NET/ROM, our software makes sure we have an AX.25 connection to that node, then puts a NET/ROM layer 3 header on our IP datagram and sends it off to the NET/ROM via an AX.25 packet with a protocol ID of NET/ROM. The NET/ROM link layer sees the protocol ID and passes the packet to its network layer, which examines its routing table and passes the packet on to the appropriate neighboring NET/ROM. This process continues until the packet arrives at the destination computer running the KA9Q software, where it is unwrapped and passed back up to the IP code.

At no point is the NET/ROM user interface or transport layer involved. We do not have to issue CONNECT commands, or make use of NET/ROM virtual circuits in any way. The NET/ROM nodes accept and pass our datagrams because they do not examine the contents of network datagrams not specifically addressed to them. We are able to take advantage of the link-layer acknowledgements and automatic routing of the NET/ROM system without the overhead of its higher level services.

SOFTWARE ARCHITECTURE

Let's examine how this is done in more detail. Our once-simple protocol stack has grown a bit by now. Let's have a look at it:

(1) The physical layer is basically unchanged (although we did added another physical layer service, described later).

(2) At the data link layer, we still have AX.25. However, the link layer now multiplexes three different kinds of packets, "No level 3" packets still go up the AX.25 session code, and IP packets will go directly up to layer 3B, but now we also direct packets with a NET/ROM PID to the NET/ROM 3A routing layer.

(3A) Incoming packets with a NET/ROM PID go to the network layer. This is a full implementation of NET/ROM layer 3. It has its own routing table, similar to that found in any NET/ROM node. It sends NODES broadcasts, which update the routing tables of neighboring NET/ROM nodes, and updates its own routing table on receipt of NODES broadcasts from those neighbors.

The NET/ROM layer examines incoming NET/ROM datagrams to see if our station is their destination. If the datagrams are not for us, the routing table is examined to see if we can forward them on to a neighboring node for handling. If we can, they are sent back down to the link layer to continue their journey. In other words, a station running the KA9Q package with NET/ROM support can act as a NET/ROM relay station. As far as neighboring NET/ROM nodes are concerned, they are simply passing traffic on through another NET/ROM.

If a NET/ROM datagram is for us, the network layer makes sure that it isn't a NET/ROM transport packet. If it is, it is dropped. If it isn't, it is sent up to layer 3B.

(3B) The intemetwork layer contains the IP router and protocol code. (Remember that IP has

its own routing table and algorithms!). It receives AX.25 traffic with a protocol ID of IP, as well as IP datagrams arriving in NET/ROM network datagrams.

The remaining layers are the same as before, so we won't repeat them.

IP ROUTING VIA NET/ROM

The IP routing table is similar, although not identical, to the one used for NET/ROM. It contains two kinds of entries, which we will call **local** routes and gateway routes.

A local route consists of an IP address and an interface name. The **KA9Q** software supports multiple interfaces, similar to the way that NET/ROM supports both a **TNC's** modem and its serial port. One component of a route, both in NET/ROM and the **KA9Q** package, is the interface through which an outgoing **datagram** should pass. (The main difference is that, while NET/ROM only supports AX.25 and two interfaces, the **KA9Q** code supports many different link layers and an almost unlimited number of interfaces.) When a local route is found in the **IP** routing table, this means that the station with the given IP address is on the local **subnet,** which for packet radio purposes means that it is within radio communications range. The **datagram** is forwarded to the link layer with an indication that direct delivery should be attempted.

A gateway route consists of an IP address, an interface name, and a gateway IP address. When we encounter a gateway route, it means that the station in question is not on our local **subnet** (i.e. not within radio range), and must be reached via a relay station, or gateway. (You will recall the idea of gateways from our earlier introduction of internetworking.) The IP **datagram** is forwarded to the link layer with an indication that the message should be sent, not directly to its destination, but to the gateway station, which will make an attempt to reach the recipient, perhaps via another gateway.

When we added the NET/ROM support, we were concerned that it be fully transparent to the IP layer, both out of concern over proper design, and out of a desire to avoid any unnecessary rewrite of the. existing code. One key assumption in the **KA9Q** IP routing software is that of routing adjacency: the IP layer makes the assumption that it can reach, via the interface given, some IP address mentioned in the route entry (either the recipient or the gateway), However, we are using NET/ROM precisely because there is no IP station within radio range who can handle our traffic. In order to maintain the adjacency assumption at the IP layer, we had to simulate the presence of an adjacent IP station in the NET/ROM code.

An IP route which uses the NET/ROM support looks just like any other routing table entry: it consists of a destination, an interface, and an optional gateway. The only difference is that the interface is called **"Netrom",** and **it's** not a link layer interface at all, although it appears that way to the IP routing code. When the IP layer sends a **datagram** down to the NET/ROM "interface" for handling, it is actually calling a small stub routine above the NET/ROM routing code. This stub looks up the IP address in a table which associates IP addresses, used at the IP layer and above, with AX.25 callsigns, used by NET/ROM% network layer. If it finds an entry for the given IP address, it creates a NET/ROM network layer **datagram** header with a destination address set to the AX.25 **callsign** found in the association table, **prepends** this header to the IP datagram, and hands it off to the NET/ROM routing code.

The NET/ROM routing software now handles the **datagram** exactly as it would any NET/ROM traffic coming in from outside: it checks to see if there is an entry for the destination AX.25 **callsign** in its routing table. If there is, it opens a link layer (AX.25) connection to the neighboring NET/ROM node advertising the best quality route, and forwards the message into the NET/ROM network, to be delivered (ultimately) to the station whose AX.25 address is that in the destination field of the NET/ROM network header, and whose IP address was that of the destination or gateway in the IP routing table entry.

This approach was extremely successful. Not one line of code needed to be changed in the IP routing code of the original **KA9Q** package.

FEATURES TO SUPPORT NET/ROM PACKET SWITCHING

As implied above, the **KA9Q** package allows an almost unlimited number of interfaces to be used for receiving and forwarding packets. At the IP layer, datagrams are routed from interface to interface using information from the IP routing table. We added a similar functionality to the NET/ROM routing layer, allowing it to receive and send NET/ROM traffic on any AX.25 interface it is configured to use. This feature allows the **KA9Q** software with NET/ROM support to be used as a multi-port full duplex NET/ROM packet switch, using standard **TNCs** or modem boards available for the IBM PC. This is vastly superior to the practice of wiring together several NET/ROM **TNCs** with a diode bridge. There is no possibility of collisions, since each TNC has its own serial port or bus address, so the interfaces can all run at full duplex, full speed through the switch. In addition, this arrangement can be used with the high-speed interfaces and modems now becoming available, far exceeding the capabilities of a standard TNC.

In several places in the United States, excess bandwidth on commercial data links is being used to carry **NET/ROM** traffic. These **"wormhole"** links work fairly well when there is only one NET/ROM at each end, but their performance degrades quickly if more are added. Beyond the difficulties inherent in the diode bridging scheme shown in the NET/ROM manual, there is an additional problem not amenable to a simple hardware solution. Most of the data links being used are running through time-domain or statistical multiplexing hardware, or through public data networks. While all of these provide some kind of carrier detect indication, in almost every case that indication comes far too late to avoid collisions. Carrier sense simply doesn't work, since the carrier indication isn't there

when it is needed, and arrives just in time to cause unnecessary delays afterwards. Performance of such an arrangement is likely to degrade to below that of schemes using no carrier sense at all.

NET/ROM nodes use a simple serial framing method to communicate with each other over their serial ports. We have added support for this framing method alongside the "KISS" protocol which the KA9Q package normally uses to communicate with TNCs. It is possible to plug a number of NET/ROM TNCs directly into the serial ports of an IBM PC, and use the PC as a switch. Some of those NET/ROM TNCs can be at the ends of "wormhole" links. These links can run at full duplex with no collisions, thus getting maximum performance and almost zero retries (assuming reliable data lines and serial interface hardware). The NET/ROM serial interface code is instrumented to provide statistics on traffic volume and error rates on its serial ports.

LESSONS LEARNED

The creation of the NET/ROM code has provided some interesting lessons on how we should and should not go about building amateur packet networks. One of these lessons became apparent before we even thought of writing the NET/ROM code.

We would probably not even have added the NET/ROM layer three support to the KA9Q package had there been an easier way to accomplish what we wanted, which was to use NET/ROM networks to handle our IP traffic until we could build our own IP network. Unfortunately, the NET/ROM software has a rather unfriendly link layer multiplexor. It sends "no layer 3" packets to the application layer, and "NET/ROM" packets to the NET/ROM network layer, but anything else it consigns to oblivion. We could have built fairly simple code to establish connections and send our IP traffic over NET/ROM transport circuits, but any packet with a protocol ID of "IP" was simply dropped by the NET/ROM software. So, lesson number one:

If you're going to build a networking product, write the multiplexing code to be inclusive, rather than exclusive. In other words, if you get something with an unfamiliar protocol ID, wrap it up and send it on, remembering to regenerate the PID properly on the other end.

Another problem we encountered was the lack of a protocol ID field in the NET/ROM network layer header. Both AX.25 packet headers and IP datagram headers contain a field which indicates what sort of higher level protocol stuff is packaged inside. This is not unlike the cans on your grocer's shelf: without a label, you have a hard time telling the beets from the beans. The AX.25 protocol ID field makes link layer multiplexing possible, and the protocol ID of an IP datagram header allows many higher level protocols to use its internetworking services. Because the NET/ROM network header does not contain a protocol ID field, there is no straightforward way to put anything but a NET/ROM transport packet inside. This is unwise. The authors of NET/ROM may well have been unaware that anyone would ever attempt a project such as ours, but by leaving this feature out of their network layer, they made it difficult, if not impossible, to ever introduce other transport protocols into their product line. So, lesson number two:

Include a protocol ID field in your link and network layer headers, even if you can't think of a use for it yet. Make it big enough to be useful, and offer to be the repository of assigned PIDs, so that a standard develops.

In experimenting with the auto-routing code in our NET/ROM network implementation, we discovered something that is a common complaint among NET/ROM operators. This can be summed up by the dictum, "Just because you can hear them, doesn't mean they can hear you." It is not unusual to have neighboring nodes that are "alligators" (big mouth, tiny ears) or for your node to be a "rabbit" (big ears, tiny mouth). Also, band openings on two meters happen quite often, and usually last just long enough for you to receive a routing broadcast from a station from whom you will never hear again -- at least until the next band opening. Either situation leaves your routing table cluttered with impossible routes, which can lead to repeated link-layer retries and transport layer failures. Routes based on band openings age out fairly quickly. Ones based on deaf neighbors come back, again and again.

After a bit of experience with this phenomenon, we added the "nodefilter" feature to our implementation. The user may specify a list of nodes which are the only ones from which route broadcasts will be accepted, or alternately, may specify a "reject list" of nodes whose broadcasts will be routinely ignored. The lesson:

If your routing method involves broadcast in an asymmetrical or inconsistent communications environment, provide a way to restrict the routes accepted to those offered by reliable nodes.

The implementation described was actually the second one we did. The first one grabbed AX.25 interfaces away from the IP part of the KA9Q code, and could only be used for NET/ROM and regular AX.25 traffic. This appeared to be a horrible idea from the moment the first version was completed, and prompted an immediate rewrite, producing a program that could act as a packet switch for IP as well as NET/ROM. The lesson here (besides "look before you leap") is:

If you're going to build a packet switch for amateur use, support link layer multiplexing, and try to make it multi-purpose, This is a hobby, and the radios, tower space, and dollars are in short supply. The more stuff you can do with a single one of each, the happier you will be in the long run.

EXPERIENCE

At this writing, experience with the software is necessarily limited. It is only now being made an official part of the official KA9Q release (thanks, Phil!), so it has not been widely available as for as long as we would have wished.

Still, it has found its way into enough hands for us to have some preliminary measurements and impressions.

The Madison, Wisconsin NET/ROM node (MAD) is connected to a node (MQTA) in Marquette, in the Upper Peninsula of Michigan, via a multiplexed commercial data line. We have conducted tests between W9NK, in Madison, and KV9P, in Alpha, Michigan. Both stations sent periodic routing broadcasts to announce their presence to their local nodes, The path chosen by NET/ROM was:

W9NK <-> MAD <-> MQT <-> MQTA <-> IRN <-> KV9P

where MQTA and IRN were NET/ROM or TheNet nodes in Marquette and Iron Mountain, Michigan, respectively, All nodes except the two using the data line were on two meters, with a speed of 1200 baud.

Performance was surprisingly good, with TCP round trip times settling in around 12 to 20 seconds, with a standard deviation of about nine seconds. In spite of the number of hops, performance was good enough to hold fairly coherent keyboard-to-keyboard conversations.

We did note at least one case where duplicate copies of datagrams were delivered by the NET/ROM network. Since TCP discards duplicates, this causes no problem in normal operations, but in the case we noticed it resulted in two replies to the same ICMP Echo Request message (produced by the ping command).

Feedback from other users, particularly NOAN in Iowa, illuminates a serious problem with the management of existing NET/ROM networks: the routing tables of these networks are so inaccurate that many experienced users don't use the network layer facilities at all! BBS mail forwarding scripts are set up to establish connections to the local NET/ROM node, request a transport connection to a selected neighbor, then from that neighbor to another, and so forth to the NET/ROM node in their destination area. They have discovered that, without human intervention, many NET/ROM networks* routing facilities break down and become unusable.

This problem has some impact on normal operations, in the sense that these multiple transport sessions do not in any sense add up to end-to-end protocol support. There is in fact no transport facility (as we understand the term) in use in these cases, since no acknowledgements travel from one end of the communications path to another. There may as well be no transport layer in NET/ROM under these circumstances; the overhead would at least be substantially lower, with no additional loss of reliability.

Unfortunately, networks in such a pathological state are unusable by our TCP/IP software. Since we make no use of the NET/ROM transport layer, we must rely entirely on the accuracy of the network layer routing tables to support the forwarding of our packets to their destinations. If these tables are not correct, our traffic will not get through.

The good news is that, in some areas where the NET/ROM operators are also working with TCP/IP, this problem is forcing them to pay attention to the quality of their routing tables. As we have noted above, NET/ROM is somewhat short on facilities to do this, but a few things can be and are being done with the tools available. One side-effect of the TCP/IP NET/ROM support may be an improvement in quality of service to all NET/ROM users!

FUTURE DEVELOPMENT

We hope, at some point, to produce a version of this support that can be put into ROM and used in a dedicated packet switch for hostile environments. Such a switch would allow us to begin building IP networks, while also offering superior performance to the NET/ROM community. It is our hope that the two user communities can work together, sharing resources to build a better network than either could alone.

ACKNOWLEDGEMENTS

Sincere thanks are due to Phil Karn, KA9Q, who would have been rich (or at least, richer) by now if he hadn't been dedicated to improving the state of the art for all radio amateurs. Also, thanks to everyone on the tcp-group mailing list, who contributed helpful comments during the design and development of the NET/ROM code. Special thanks also to Howard Leadmon, WB3FFV, and John Limpert, N3DMC, who got it to compile under Unix.

Individuals who helped test or provided feedback on experience with the software were: Duane Brummel, NX9K; Hasan Schiers, NOAN; and Dave Reinhart, KV9P.

Phil Karn, KA9Q, reviewed the first draft of this paper, providing many helpful comments and suggestions.