Amateur TCP/IP: An Update

Phil R. Karn, KA 9Q

1. Introduction

Amateur radio use of the DARPA Internet protocols (''TCP/IP'') has grown from a paper proposal during the "protocol wars" of several years ago to a well-established reality today. Because the TCP/IP software is free and available to radio amateurs and all other noncommercial users, it is hard to say exactly how many are using it. One rough estimate is the number of Internet addresses that have been assigned from the "network 44" block for amateur packet radio: about 1,000 amateurs in several dozen countries. The package has also gained considerable popularity outside of amateur radio, especially in universities.

With the popularity of TCP/IP on amateur radio has come another most welcome development: the appearance of others making substantial contributions to the software effort by creating new features and enhancing existing ones. Several of these contributors have documented their work in other papers in these proceedings, and any other potential contributors are also encouraged to do so. In this paper I will review the TCP/IP developments and experiments of the past year. Although I will mention several contributors by name, the project has grown much too large for this to be an exhaustive list; I hope no one will feel slighted if they are accidentally omitted.

In this paper I will also comment on some of the lessons learned so far, and then discuss possible directions for the future. As expected, much has been learned about the operational aspects of true computer networking on amateur packet radio. We've also learned quite a bit about coordinating the development of a complex software package when volunteers all over the world are involved.

2. New Features) and Enhancements

2.1. AX.25 Support

A complete AX.25 Level 2 implementation has been added to the package. Its primary purpose is to provide hop-by-hop acknowledgement of IP datagrams without having to rely on TCP for end,-to-end retransmission. IP datagrams may now be carried either in connected-mode I frames that are acknowledged at the link layer, or in AX.25 UI frames as before. The default encapsulation mode is set in the configuration file. Individual datagrams can override the default with the type of service (TOS) bits in the IP header.

An optionall "transparent fragmentation" facility breaks up large IP datagrams into a series of AX.25 I frames for transmission over poor links without added TCP/IP overhead. This was implemented as a local extension to the AX.25 protocol.

AX.25 can also be used directly from the keyboard (i.e., without TCP/IP) for communication with ordinary packet stations. Because of the multiplexing provided by the AX.25 Protocol ID byte, "conventiona.!"' AX.25 and TCP/IP/AX.25 operation can take place simultaneously, even between the same pair of stations.

With minor exceptions, the AX.25 code tracks the proposed AX.25 Version 2.1 specification currently under review by the ARRL Digital Committee. The code has been recently rewritten to adhere to the SDL diagrams by K3NA as closely as possible.

2.2. IP-on-NET/ROM

A separate paper in these proceedings by Dan Frank, W9NK, documents an important contribution to the package: the ability to pass IP datagrams through NET/ROM networks.

This is very much in keeping with a fundamental principle of internetworking that accounts for much of TCP/IP's success: the ability to use facilities that were designed by others without internetworking in mind. Dan's paper discusses in detail the specific approach taken and its advantages and disadvantages.

2.3. Packet Driver Support

FTP Software, Inc, has released into the public domain its interface specification for packet-oriented hardware device drivers on the IBM PC. Packet drivers run as Terminate and Stay Resident (TSR) modules. They are loaded independently of the protocol modules that use it (e.g., the net.exe program that implements TCP/IP on the PC), and they can be shared by multiple protocol modules under a multitasker such as DoubleDos or DesqView.

A major practical advantage of the packet driver approach is that the driver software can be developed and maintained completely independently of the protocol code. Since the drivers no longer need reside in net.exe, the latter can be much smaller; users need not waste disk space or memory on drivers for devices they don't have.

As the packet driver specification becomes a de-facto standard, we will be able to use drivers written by others. TRW had already contributed its PC-2000 Ethernet card driver when I added packet driver support to net.exe. Russ Nelson has since written packet drivers for the Interlan NI5120 and 3Com 3C501 Ethernet cards, and Bob Clements, K1BC, has written one for the Western Digital WD8003 Ethernet card.

2.4. Other Drivers

Art Goldman, WA3CVG, and Richard Bisbey, NG6Q, have contributed a driver for the Eagle HDLC interface card. This card, for a time a popular item at swapmeets, features the Zilog 8530 chip. They started with an 8530 driver I wrote for the PACCOMM PC-100 and added many performance enhancements.

2.5. Routing

Al Broscius, N3FCT, has written automatic routing code for the package. The protocol is the widely-used "RIP" (Routing Information Protocol) that has been the informal standard for several years with Berkeley UNIX systems. (Berkeley adopted it from XNS, the Xerox Network System.) RIP belongs to the class of routing algorithms known variously as "Bellman-Ford," "Distance Vector," or "The Old ARPANET Algorithm." NET/ROM's internal routing algorithm is also in this class, although it differs in detail from RIP. Al's contribution is intended primarily for Ethernet LANs where RIP is already common.

2.6. Electronic Mail

The original mail subsystem by Bdale Garbee, N3EUA and myself has been greatly extended and enhanced by Dave Trulli, NN2Z. Dave and Bob Gibson, WA3PXX, have also each implemented automatic SMTPIWORLI mail forwarding systems. This is a particularly useful feature as it allows those running TCP/IP systems to receive "regular" packet mail without having to log into a BBS manually.

2.7. Additional Applications and Features

Several useful applications have also been contributed. Mike Horne, KA7AXD, has written a "finger" server. This is a popular ARPA application that allows one to locate ("put the finger on") a person on a remote system. It is a handy way to make information such as your telephone number or electronic mail address available for the convenience of those wishing to contact you. W9NK has also written an AX.25 "mailbox" that makes the package more useful with conventional packet stations.

I have added several simple features that make it easier to run the code at a remote site. A "remote" server and command allow control stations to reboot the system with new software and/or configuration files, and a "forward" command handles simplex links such as those found in the collision free backbone node described later.

3. Internal Changes

In addition to the new externally visible features that have been described, quite a few changes have been made under the surface. While perhaps not of direct interest to those who simply use the package, these changes are important because they improve performance, reliability and portability, or make it easier for programmers to create new applications.

3.1. Congestion Control

A major contribution to the problem of avoiding congestion in a highly dynamic TCP/IP Internet has been made in the past year by Raj Jain of Digital Equipment Corporation and Van Jacobsen of Lawrence Berkeley Laboratories. Van's suggestions have been widely accepted by TCP implementers, and they are included in my TCP code. I have also contributed to this effort by devising a technique that guarantees the correctness of the round trip time measurements that TCP uses to set its retransmission timer.

3.2. Improved CPU Portability

The ordering of bytes within a machine word, the hardware alignment requirements for various data types, and the exact positioning of elements in a C-language structure all depend on the specific processor and compiler used. These differences can create portability problems. For example, early ports of the package to the Motorola 68000 sometimes generated faults due to the 68000's requirement that short and long integers be at even addresses, something not required by the 8088.

To eliminate these problems, the subroutines that generate and process protocol headers were rewritten to be completely independent of these considerations. Each protocol module contains two conversion routines: one that converts the protocol header from the external (network) format specified by the protocol standard to an internal representation, and another for the inverse function. External representations are always kept as a simple byte string, while the internal representation is usually a C-language structure. With this and other changes, portability to processors other than the Intel 8086 family (used in the IBM PC) has been greatly improved.

The package necessarily contains some 8086 assembler code, despite its inherent nonportability. The assembler code is in three categories:

3.2.1. Interrupt Handlers

Since the various device drivers are interrupt driven, an assembler "stub" fields each interrupt. Each stub establishes a C environment and then calls a C-language function to do the "real" work in handling the interrupt. Although some newer compilers (e.g., Turbo C and Microsoft C) support special "interrupt" functions, they appear to be primarily designed for software interrupt handlers; extra code is still required (either inline or regular assembly code) to make them function as hardware interrupt handlers.

3.2.2. DOS Interfaces

Some assembler code is included because the package uses several DOS system calls that are not supported by the standard vendor-supplied C library. While C passes function arguments on the stack, DOS and BIOS functions accept their parameters in registers; hence the need for special assembler-language interface functions.

3.2.3. Performance Enhancements

Two functions were: written in assembler, even though they had already been done in C. One routine computes the ones-complement sum of a block of 16-bit integers; this is the ARPA standard checksum algorithm. The other does fast I/O to port-mapped devices such as the 3Com 3C501 Ethernet controller. Writing these routines in assembler improved performance through the use of certain 8086 instructions that are unavailable directly in C. For example, although the checksum algorithm operates on 16-bit integers, the C implementation uses long (32-bit) addition to accumulate the end-around carries. 32 bit arithmetic on the 8086 is relatively expensive because the CPU is a 16-bit unit. However, in assembler the ADC (add with carry) instruction makes it possible to sum each 16-bit word with only two instructions: a LODSW to fetch the word from memory and an ADC to add it plus the carry from the previous addition to the running checksum. Loop "unwinding" further enhances performance.

I usually try to avoid assembler code when possible because of its inherent nonportability. The problem isn't so much in porting to machines completely different from the PC; device drivers tend to be (necessarily) hardware dependent, so they have to be rewritten anyway. The main headache has been nonportability to the various C compilers on the PC. For example, Aztec C provides a macro package that makes it very easy to write C-callable assembler routines. These macros automatically "do the right thing" for each of the 8086's "billyuns and billyuns" of memory models. Unfor-

tunately, a compatible set of macros isn't available with the other compilers, so Russ Nelson contributed quite a bit of time and effort in porting the assembler routines to Turbo-C.

3.3. Improved Operating System Portability

We have tried to improve portability to operating systems and compilers other than MS-DOS and Aztec C. We found this to be much harder than simple CPU portability since file system, native I/O device support and C subroutine library dependencies appeared in widely scattered parts of the package. For example, while the Apple Macintosh, MS-DOS and UNIX all provide hierarchical file systems, the Mac uses the colon (:) to separate path name components while UNIX uses the slash (/) and MS-DOS the backslash (\). Although many of these problems quickly became obvious once a porting effort had begun, it was unexpectedly difficult to identify them in advance. A typical porting effort therefore goes through several iterations with the goal of isolating as much system-dependent code as possible to a small number of files. We have partially succeeded in this, but there are still many #ifdefs (conditional compilation segments) scattered throughout the code. This should improve as the package matures further.

Porting has involved the efforts of several people. Mikel Matthews, N9DVG, did the initial ports to the Apple Macintosh, UNIX System V and the Commodore Amiga. A group in the San Francisco Bay area including Dewayne Hendricks, WA8DZP and Andy Cromarty, N6JLJ has made further improvements and additions to the Macintosh version, and Bob Hoffman, N3CVL is now coordinating changes to the UNIX version. There are undoubtedly others working on porting efforts of which I am unaware.

4. Operational Experience

Most on-air TCP/IP operation to date consists of local "islands" of activity that cover metropolitan areas or regions. The facilities used in each area are diverse; some use IP switches almost exclusively, others use various combinations of analog (FM) repeaters, IP switches, AX.25 digipeaters and NET/ROM nodes. Most activity uses the 1200 baud AFSK format, simply because the equipment is so widely available. The GRAPES group in Atlanta has pioneered the use of the 56 kbps modem by Dale Heatherington, WA4DSY, and TCP/IP has been its primary application.

As expected, the Internet approach has proven highly versatile in adapting to these heterogeneous environments. Once a system has been properly configured, its user merely issues network commands that specify the system with which he wishes to communicate; he need not be continually concerned with the idiosyncrasies of his local network.

4.1. Collision Free Networks

The "collision free backbone" technique described at last year's conference has been implemented in an experimental node in northern New Jersey. Our site receives on two UHF frequencies (431.025 and 440.950 MHz) and transmits on 221.07 MHz; crossband operation allows it to receive and transmit simultaneously. This technique works, although good performance still requires good RF links. It is interesting to hear this site in operation for the first time; instead of the short bursts of useful data interspersed with collisions, idle flag streams and squelch tails one is accustomed to hearing on regular packet channels, the switch's carrier can stay on continuously for minutes at a time during a file transfer, relaying back-to-back data packets and acknowledgements.

4.2. Radio/Wire Internetting

Some experimental linking between different TCP/IP "islands" has been done with GTE Telenet's PC Pursuit service. This inexpensive service provides slow speed (1200 baud) asynchronous communication across Telenet's X.25 network during nights and weekends when excess capacity is available. Since PC Pursuit provides a (logically transparent) asynchronous "bit pipe" between its endpoints, it handles SLIP (Serial Line IP) just fine. A station in each area acts as a gateway, relaying packets between the local packet radio channel and the telephone line. When more than one telephone line and modem is available at a given station, that site can act as a "hub", switching datagrams from one phone line to another as well as between phone line and radio channel. These experiments clearly demonstrate an ability to establish an emergency packet switching network out of ad-hoc combinations of telephone lines and radio channels.

4.3. Politics

Certain TCP/IP groups have unfortunately encountered occasional friction, especially in areas where TCP/IP activity coexists with conventional AX.25 activity. The complaints seem to fall into two categories: channel loading and "garbage characters", but neither problem is unique to TCP/IP. Channel loading is a potential problem wherever interactive traffic competes with file transfer traffic for a low speed channel. One feature of the TCP/IP package that alleviates this problem somewhat with respect to regular AX.25 users is that TCP "backs off" exponentially whenever it loses a packet, i.e., it doubles the time between each successive attempt. This makes TCP far less aggressive on a heavily loaded channel than a regular AX.:25 TNC.

Complaints about "garbage characters" occur when the binary headers of IP or TCP are seen by stations passively monitoring the channel with regular TNCs. Such complaints are not limited to TCP/IP; any high level networking scheme, including NET/ROM, has similar problems, as do binary file transfers. Clearly the correct solution lies with more flexible packet monitoring code in the TNCs, as enough information is available in each AX.25 packet header (specifically in the PID field) to determine what higher level protocol is in use.

5. Work in Progress

This section discusses some of the ideas and goals for the package that are currently being thought about or implemented. These cover a variety of items, both hardware and soft ware.

5.1. Multitasking operating system

Net.exe has long been structured as a "commutator loop" that relies on upcalls and polling. Hardware interrupts simply transfer data between devices and queues that are operated on by routines called from the main loop. (See my earlier papers for further details).

Each application presently provides functions (entry points) that are called asynchronously by the transport protocol modules when external events occur. Applications are not permitted to "busy-wait", and they must explicitly save state in "control blocks" so that each call is interpreted properly. This technique worked surprisingly well for the existing applications (the DARPA FTP, Telnet and SMTP protocols) but it can be clumsy.

With many expressing interest in using the package as a base for more complex applications (e.g., multi-user WORLI-style bulletin boards) I have begun to restructure the package around a multi-tasking kernel. This will provide a simpler programming environment more like that of a conventional multitasking system. Net .exe will remain a single executable file containing all of the applications linked together, and it will still run under a higher level multitasker such as Desqview or DoubleDos.

New "synchronous blocking" primitives more like those in conventional multitasking operating systems will be provided. For example, calls to the network "receive" function will block, if necessary, until data is available. When one task blocks, others (if any) will be run automatically.

The operating system kernel will be non preemptive. That is, once a task gains control it runs until it explicitly gives up the processor by executing a "wait" primitive (e.g., inside the tcp_read subroutine). A CPU-intensive task can also voluntarily relinquish the processor, allowing other tasks to run. without actually waiting for an event. Choosing a non-preemptive kernel greatly simplifies the design of the rest of the system, since "critical sections" are almost eliminated. (Critical sections are those sections of a program where a hardware interrupt followed by a task switch would leave data structures in an inconsistent state, usually causing a system crash. Critical sections can be notoriously hard to find, since interrupts are semi-random external events).

Most multitasking kernels require some assembler language code for saving and restoring a task's registers but the non-preemptive approach combined with a very clever trick suggested by Rob, PE1CHL allows this to be done with the standard C library's setjmp/longjmp mechanism. This technique makes the resulting code much more portable, thereby avoiding the main reason I decided against a multitasking kernel at the beginning of the project.

5.2. Where Is My High Speed Digital?

TCP/IP users feel the limitations of 1200 baud much more acutely than most packeteers, mainly because of the extreme ease with which

parallel operations and large file transfers can be commanded. The problem is much like that of running UNIX or some other powerful timesharing system on a PC/XT. Even though a specific program might run just as fast as under a single-user system like MS-DOS, the system quickly runs out of gas when the much more powerful features of the timesharing system are exercised.

Dale Heatherington, WA4DSY, has made an enormous contribution toward solving the raw RF link bandwidth problem with his 56 kilobit MSK modem design. I have built several of these modems, and they certainly work as advertised. On Ethernet, the TCP/IP package transfers a file between a pair of IBM PC/ATs at about 250 kilobits per second. This is by no means blazingly fast by Ethernet standards (the code was written for portability and flexibility, not maximum speed) but it should certainly be able to keep a mere 56 kilobit modem occupied. But Ethernet interfaces are fairly smart devices designed for fast transfers; unfortunately there is as yet no readily available serial interface for the WA4DSY modem with the capacity these modems provide! (Those who saw the original Dayton announcement of the WA4DSY modem may remember its subtitle: "Where, Oh Where Is My High Speed Digital?")

The standard asynchronous serial ports found on personal computers cannot be used directly with Dale's modem as it is synchronous (as is regular 1200 baud packet). The Atlanta group has had moderate success by modifying standard TNC-2s to operate at 56kbps on the radio side, but they are still limited to relatively low data rates (e.g., 19200 bps) between the TNC and the host computer. This prevents full use of the modem's capacity.

It is better to eliminate the TNC entirely and use an interface card on the PC to generate HDLC that can be fed directly to the modem. Several HDLC cards already exist, including the HAPN 8273 card, the PACCOMM PC-100, the DRSI PCPA and the aforementioned Eagle card. However, none of these cards work well at high speeds, because the PC's DMA (direct memory access) facility is insufficient to support these cards without additional hardware. Interrupt driven transfers are entirely appropriate at lower speeds, but at 56 kbps a byte arrives every 142 microseconds; that is not much time on a 4.77 Mhz 8088 to save state, process an interrupt and return. Making matters much worse is that much PC system code disables interrupts for relatively long intervals (consider disk transfers), effectively "starving" a hungry high speed modem.

It is possible to make one of these simple cards operate in half duplex at 56 kbps -- after a fashion. I am presently writing a driver for the DRSI PCPA that will run at 56 kbps, but it requires that interrupts be inhibited during any active packet transfer. Packet transmission and reception are both done with "busy waits"; the transmit routine is invoked whenever a packet is to be sent while the receive routine is invoked by the appearance of a receive carrier. (This has an unfortun ate side effect: open the demodulator squelch and the system "hangs" until the squelch closes).

The proper long-term solution to the "Where Is My High Speed Digital" problem lies with hardware: a special "slave" processor board with sufficient intelligence, buffering and autonomy to handle several seconds' worth of packets without immediate assistance from the host processor. Mike Chepponis, K3MC, has been working on the design of such a board, and his paper on this subject appears in these proceedings. His design appears sufficient for speeds considerably faster than 56 kbps as well.

5.3. Routing

Saying that automatic routing is a "fertile research area" in the professional computer science community is like saying that Antarctica has a lot of ice. Much practical experience has been gained from many commercial, military and research computer networks that can now be applied to amateur packet radio, and I would like to offer some thoughts on how the static, manually controlled routing in the TCP/IP package might be replaced.

As mentioned in the section on RIP, many existing networks, including NET/ROM, use a routing algorithm that is variously known as "Bellman-Ford", "Distance Vector" or "The Old ARPANET Algorithm". Each node broadcasts its routing table, the entries of which list each destination in the network and that node's current estimate of the "cost" to that destination. This algorithm is known to have many problems in practice, particularly with susceptibility to routing loops when links fail. Radio creates a few problems of its own, includ-

ing particularly flakey links and non-reciprocal paths.

A different algorithm, known variously as "Dijkstra's algorithm," "Link State Routing," ''SPF'' or "The New ARPANET Algorithm" may offer an (alternative. In link state routing, each node broadcasts to all other nodes the status of each of its local links. Each node receives the broadcasts of the other nodes and constructs a complete map of the network from which it may make local routing decisions. The broadcast mechanism is in fact implemented with "intelligent flooding", where an incoming packet is relayed by a node to each of its neighbors only if that packet hasn't been previously seen. (This mechanism has other uses, such as the provision of a user broadcast mechanism).

The only special problem radio presents to this algorithm is how to determine whether a link is "up" or "down". Just having an active AX.25 connection isn't enough; it might have been idle for several hours, so there's no way to know if it's still good or not. Continually transmitting link test packets ("pinging") is not a particularly good idea, for fairly obvious reasons. Here an idea from the DARPA packet radio projects should be very helpful. Each node only has to broadcast a periodic report listing the stations it has heard in the last N minutes, with a count of the number of packets seen from each. Each station mentioned in the report can then compare that number against the number it actually transmitted, gauging the reliability of that particular path. If the percentage is high enough, it can consider the link to be "up"; otherwise it is "down". Or intermediate figures representing link "quality" could be used. Clearly there is plenty of interesting and useful work that can be done here.

6. Concluding Thoughts

Bringing TCP/IP to amateur radio has been a very time-consuming task, but its acceptance has been a most gratifying reward. Many challenges still remain, and not all of them are technical. We have: learned and demonstrated many things so far. We have proven that TCP/IP is indeed a practical and versatile foundation for the more advanced amateur packet applications, but we have also learned that many other packeteers do not as yet feel that they need its capabilities. There now seems to be a growing awareness within amateur packet radio that, at present, no one approach satisfies everyone's needs and that there is plenty of room for parallel, complementary approaches. As more powerful hardware becomes popular, though, I do believe that the "computer networking" philosophy as demonstrated by the TCP/IP project will gradually replace the more traditional "terminal networking" systems we have now.

The KA9Q Internet package continues to be available on the same terms: it may be freely copied, used and modified for amateur or other noncommercial use. Commercial use of any part of the package requires permission of the appropriate author or authors.