The KA9Q Internet (TCP/IP) Package: A Progress Report

Phil Karn, KA9Q

ABSTRACT

For over two years, the author has led the development of C-language software implementing the ARPA Internet protocol suite (commonly called "TCP/IP"). Intended for amateur radio use, the software was originally written on and for the IBM PC and its clones running MS-DOS. However, the use of a de-facto industry standard protocol set has resulted in considerable interest in and contributions to the effort by non-amateurs as well. The software has been "ported" to several different computers and is enjoying increasing use in both conventional Local Area Network (LAN) environments as well as amateur packet radio.

This paper describes the considerable progress this effort has made, and reflects on the choice of TCP/IP now that significant on-air experience has been gained.

1. Introduction

At the Fourth ARRL Amateur Radio Computer Networking Conference in March 1985, I proposed that the ARPA Internet Protocols be used in amateur packet radio. [2] [3] Since then, TCP/IP has become a reality on amateur radio. Many stations now run TCP/IP almost exclusively, and it is increasing steadily in popularity.

2. What "TCP/IP" is - and isn't: A Review

Unfortunately, the subject of higher level networking protocols in amateur radio (including, but not limited to, the famous virtual circuit vs datagram debate) is still highly controversial in certain circles. Because many misconceptions about TCP/IP persist, I will review what it is and is not.

TCP and IP are only two elements (albeit very important ones) of a larger, modular set of protocols known as the ARPA Internet Protocol Suite. It must be stressed that only "higher level" protocols are specified. In ISO jargon, the ARPA Protocol Suite begins with the upper half of the network layer (level 3B, also called the internet sublayer) and goes all the way up to level 7, the application; levels 1, 2 and 3A are deliberately left unspecified. This is in keeping with ARPA's original purpose: constructing a uniform *internetwork* out of an existing collection of dissimilar links and even entire networks that would otherwise be incompatible with each other. There is no "ARPA Standard Link Level Protocol" because there is no need to require only one; they can all coexist yet still interoperate.

AX.25 Level 2, X.25 and the network layers of NET/ROM, Texnet, and even COSI fit into the Internet model very nicely. TCP/IP does *not* compete with these developments (except for the level 4 or transport level components in some of them), but rather *complements* them all. By filling a very real need, the ARPA suite is today *the* de-facto industry standard for computer networking when a wide variety of computers and underlying networking technologies must be interconnected.

3. What "Higher Level Networking" Really Means

In the ISO model, everything above layer 3 is "end-to-end". That is, layers 4 through 7 exist *only* in the end users' machines *(hosts* in ARPA terminology, *end systems* in ISO) not in the intermediate packet switches *(gateways).* Unfortunately, some have used terminology at variance with this definition, causing considerable confusion as to the meaning of "higher level networking". For example, the addition of hop-by-hop acknowledgements to a network is *not* "level 3 networking", it is a level 2 function. Further, by strict interpretation of the ISO model, we already have a de-facto datagram-oriented

"network layer" in the address field that is part of "AX.25 Level 2". We already have a de-facto "transport" protocol running in the TNCs that maintains connections on an end-to-end basis. Installing "real" level 3 and 4 protocols means pushing AX.25 back down to the link layer it was designed for.

These distinctions are not just semantic quibbling. They affects one's entire view of how the pieces should fit together in the network and how it should appear to the users.

4. Computer Networking vs Terminal Networking

It should now be apparent that the purpose of higher level protocols is to connect *computers,* not just terminals, and to use the capabilities of those computers effectively. (Running a "terminal program" on a PC is *not* using it effectively.) True networking is much more than just providing "virtual wires" between dumb terminals[1] or even a dumb terminal and a bulletin board system. While packet's use of faster modems, addressing, error detection and retransmission does provide channel sharing and error-free communications, without higher level protocols running on the user's computers the result is *qualitatively* little different than ordinary radio teletype (RTTY).[2] In contrast, *true* networking involves one or more high level (application, presentation and transport) protocols plus a multi-tasking operating system running in the end-user's machine, performing end-to-end functions automatically on behalf of human users. For example, a single system might respond automatically to remote requests for file, accept remotely sent files and electronic mail for storage, and initiate similar operations on other computers in response to local commands -- all simultaneously, with minimal operator intervention. In the following sections I will review the major ARPA protocols and describe the implementation of those in the KA9Q Internet software package, starting with the top layers and working down.

5. Internet Software Components

A major goal of the KA9Q Internet package was that multiple network activities should go on simultaneously. This greatly increases the usefulness of the system, and alleviates many other potentially troublesome problems. For example, "stuck connections" caused by the other station abruptly disappearing are only a minor annoyance, as the only resources wasted are a few bytes of RAM. The system can still be used by others; keepalive timers aren't necessary.

Since MS-DOS is not a multitasking system, all of the protocols were combined in a single MS-DOS program called **net.exe,** and a very simple form of multitasking is performed internally by a "commutator loop" mechanism.[3] The main loop of the program sequentially polls various routines to see if they need service. For example, the keyboard is checked for input, then the serial input buffer is checked for characters, then the Ethernet receiver is checked for packets, then the timer is checked to see if a tick occurred, and so on.

When events occur, calls are made to the appropriate routines; incoming data triggers the appropriate link level protocol modules, which in turn call the higher level modules, and eventually the applications are called. This is known as an *upcall* or *pseudo-interrupt* mechanism, and has been used for other small networking packages such as MIT's PC/IP. It is important to note that there is no conventional sleep/wakeup mechanism; each application must provide functions to be called asynchronously by the system. These functions cannot block or hog the processor; they must respond to the event and return. The applications must therefore be structured as state machines driven by upcalls. While this is a somewhat unusual environment for a programmer, it isn't too hard to get used to it. In fact, it encourages the programmer to think about what should happen for every possible combination of state and event; it's easy to get sloppy about this in a conventional environment by assuming that only the desired event can occur in a certain state.

---

[1] The term dumb terminal was originally a trademark of Lear Siegler Corporation for their ADM-3 CRT terminal. The term quickly became generic, referring to any keyboard/display (or printer) combination lacking programmability and local file storage. Only recently has it really become pejorative, since many complete personal computers now cost significantly less than dumb terminals.

[2] Some call conventional packet operation "RTTY packet".

[3] This is somewhat similar to the multitasking form of FORTH known as IPS, or Interpreter for Process Structures. IPS was developed by DJ4ZC for use in the AMSAT Phase 3 satellites.

I do not mean to imply that the commutator loop approach is superior. I originally chose it as a simple expedient, and since then I've been surprised by how much I've been able to do with a very small amount of memory. The entire executable program net.exe, containing all of the protocols about to be described, is only about 60K bytes. In comparison, the popular PC terminal program "Procomm" is almost three times as large![4] One major drawback of the upcall structure, however, is the lack of application portability. Future developments may include a true multitasking kernel so that a more conventional programming environment may be supplied as well.

The user interface is simple, but functional. Commands to invoke each of the applications are provided, along with others primarily useful for monitoring and statistics gathering. Up to ten client "sessions" may exist at any one time, and the user may switch between them at will. There is no limit on the number of server sessions that may exist at one time other than the memory available on the machine for buffering and housekeeping.

5.1. Telnet, FTP and SMTP Applications

While many application protocols have been built for packet networks, three are most useful: remote login, file transfer and mail transfer. This is reflected in the "big three" ARPA Internet application protocols: Telnet, FTP and SMTP respectively. Each application protocol is further broken down into *client* and server halves. Clients act on behalf of local users by initiating communication with remote servers that passively await their requests.

Clients and servers for all three protocols are presently in the software, although the telnet server does nothing more than route an incoming connection to the console for keyboard-to-keyboard chatting. (MS-DOS isn't a timesharing system, so it wasn't possible to provide conventional telnet service). FTP supports both ASCII (default) and binary file transfers, with passwords protecting against unauthorized file access. SMTP is presently functional but it does not have the features of mailers found on larger systems such as mailing lists and mail-level forwarding. Two miscellaneous application servers are also provided: echo and discard. They are intended mainly for testing.

5.2. TCP and UDP Transport Protocols

The applications just described all use TCP, the Transmission Control Protocol. TCP is a transport/session layer (level 4 and 5) protocol that provides virtual circuit services on an end-to-end basis. The use of TCP is not mandatory, however. Some important applications prefer not to use virtual circuits, so an alternative is supplied: UDP, the User Datagram Protocol. UDP is important for routing algorithms, information broadcasting and transaction-oriented applications such as the Network File System (NFS).[5]

The TCP was the first module written for the package and is now quite mature. Three upcalls are provided to the application layer: receive, send and protocol state change. The receive upcall indicates that data has arrived which may be read from the receive queue by the application. The send upcall indicates that outgoing data has been acknowledged, freeing up buffer space that may now be used for additional transmissions. The protocol state change upcall indicates when connections are opened and closed, and applications use these to drive their own state machines and to determine end of file. Much effort has gone into tuning the TCP retransmission algorithms for efficient operation across amateur packet radio, including a novel approach to measuring round trip times accurately during periods of high packet loss. [6]

The UDP module is much simpler than TCP. Only a receive data upcall is provided.

---

[4] I do admit that net.exe lacks certain essential features, e.g., exploding windows and sound effects.

[5] NFS is not implemented in the package (yet) so it will not be described here. However, its popularity is mushrooming around the Internet. On many Ethernet local area networks (LANs) NFS/UDP traffic now exceeds that using TCP.

5.3. Internet Protocol (IP)

The core protocol of the ARPA Internet is called, naturally enough, the Internet Protocol (IP). IP sits immediately above the existing lower level networks (*subnets* in ARPA terms). This is the only mandatory protocol in the entire suite. The higher level protocol in use (TCP, etc) need be agreed upon only by the hosts involved, but you can't be "on the Internet" unless you run IP.

Since IP is "spoken" by hosts and "interpreted" by the gateways (packet switches), I found it convenient to split my IP into two halves. The upper half contains the parts of IP relevant to a host (outgoing packet generation, incoming packet demultiplexing, fragment reassembly). The lower half contains the IP packet switching components (routing and fragmentation). Every host is also a gateway, i.e., it will route and switch traffic that is just passing through in addition to sourcing and sinking its own traffic. If the code is to be run on a dedicated gateway, the host functions can be removed to save space.

The packet routing portion of my IP includes a generalized form *of subnetting*, the ability to structure the address space into a tree-shaped hierarchy. [3] Each entry in the routing table includes a width field saying how many bits in its address field are significant. When packets are routed, the algorithm finds the routing table entry providing the "best match" to the leading bits of the destination address. This allows large blocks of addresses that share a common next hop, e.g., all west coast addresses in an east coast switch, to share a single routing table entry. This reduces the average size of a routing table enormously while still allowing arbitrary routes to be set up. Even this relatively complex technique, however, only takes about 6 milliseconds to route a packet on the IBM PC, thereby refuting the argument that datagram routing "costs" too much.

No automatic routing algorithm is provided as yet; the routes must be set up manually. Automatic routing is clearly a desirable long-term goal, but it is a difficult and challenging problem in any network so I have deferred it.

5.4. Subnet Protocols

Link/subnet drivers for AX.25 Level 2, Ethernet and asynchronous point-to-point links are presently provided. In the spirit of the Internet, others can be added easily as they become available (e.g., NET/ROM, Texnet and COSI).

The AX.25 Level 2 driver is at present very simple; each IP datagram is encapsulated in a single AX.25 UI (connectionless) frame. The KISS TNC [1] was developed to allow these "raw" packets to be generated. This was an expedient for initial operation; it is *not* the only way that IP can be run on top of AX.25. It is entirely possible to use the full-blown connection-oriented AX.25 protocol under IP if necessary to improve hop-by-hop reliability; this will have to be done to use the internals of NET/ROM to move IP traffic, for example. On the other hand, collision-free backbone channels would do well to avoid the extra complexity and overhead of link level acknowledgements.

It is incorrect to say that IP cannot be run efficiently on noisy poor channels because of its relatively large header size. While not done at present, the subnet or link may perform intranet fragmentation. That is, it may chop up a single datagram into multiple smaller packets and reassemble them transparently at the other end of the link before passing them up to IP. This is distinct from the internet level fragmentation done by IP, and is preferable for performance reasons when the subnet has an unusually small packet size limit. As for the "inordinate" overhead associated with a 40-byte TCP/IP header, consider that even at 1200 baud, 40 bytes takes only a quarter of a second to send. Many stations spend more than this just keying up the transmitter. As faster modems (e.g., the new WA4DSY 56 kbps design) become widespread, header overhead will become even more of a non-issue.

5.5. The Address Resolution Protocol (ARP)

The Internet Protocol provides its own addressing independent of that used in the subnetworks. It is therefore necessary to map IP addresses into subnet addresses for each hop. Sometimes this can be done by making the subnet address part of the IP address, but frequently this isn't possible because the subnet address is too big. This is the case with both Ethernet and AX.25, so the Address Resolution Protocol was implemented. [7]

The ARP module in the package serves both subnet protocols. Since ARP requires a broadcast facility in the subnet, I chose "QST-0" as the AX.25 broadcast address. Manual commands are provided to manipulate the ARP translation table, either to override the automatic mechanism or to specify multi-hop digipeater paths. The latter must be done manually since the AX.25 subnet can no longer provide broadcasting when digipeaters are involved.

6. Availability

The entire software package, including executable programs, complete source code and documentation, is available to interested amateurs for the cost of copying only. It may be freely used and copied for noncommercial purposes only. Brian Lloyd, WB6RQN, handles the distribution on MS-DOS format floppy disks; send him $5 to cover costs and he will provide disks, mailers and postage. Persons outside the US should add enough to cover higher postage costs.

7. Credits

Many people have contributed in various ways to this effort, so what follows is necessarily only a partial list. Bdale Garbee N3EUA wrote the mail command *bm,* and coordinates the integration of software releases. Mike Chepponis K3MC wrote the first KISS software for the TNC-2 and has been instrumental in encouraging others to support other TNCs (see [1] for a complete list). Jon Bloom KE3Z contributed the driver for the HAPN HDLC adapter card for the PC. Brian Lloyd WB6RQN has widely promoted the amateur use of TCP/IP, writing introductory magazine articles geared to the novice user. When the code first became available, Brian organized a local group of users in the Washington DC area that has since grown to several dozen; their feedback, as well as that of other groups around the world, has proven very useful in improving the package. Brian also spends a considerable amount of time distributing the package on floppy disk by mail.

While I wrote the bulk of net.exe myself, others supported my efforts by patiently answering my many questions about the details of the protocols. Thanks go to Dave Mills W3HCF of the University of Delaware and Jon Postel of the University of California Information Sciences Institute.

8. References

1. Chepponis, M., and Karn, P., "The KISS TNC: A simple Host-to-TNC communications protocol," this conference.
2. Karn, P., "TCP/IP: A Proposal for Amateur Packet Radio Levels 3 and 4," Fourth ARRL Amateur Radio Computer Networking Conference, San Francisco, March 1985, p. 4.62.
3. Karn, P., "Addressing and Routing Issues in Amateur Packet Radio," Fourth ARRL Amateur Radio Computer Networking Conference, San Francisco, March 1985, p. 4.69.
4. "For own in-house network, COS selects TCP/IP," Data Communications magazine, June 1987.
5. Bloom, J., "NET.EXE: Beyond the TNC," Gateway Vol. 3 No. 12, Feb 6, 1987.
6. Karn, P., and Partridge, C., "Improving Round-Trip Time Estimates in Reliable Transport Protocols," SIGCOMM Workshop proceedings, Stowe, Vt., August 1987.
7. Plummer, "Address Resolution Protocol," ARPA RFC 826.