TCP/IP on FlexNet - Just Another Layer

Gunter Jost, DK7WJ/K7WJ
Lichtenbergstrasse 77,D-64289 Darmstadt, Germany
Translated by: Don Rotolo, N2IRZ
c/o RA TS, PO Box 93, Park Ridge NJ 07656 USA

Abstract:

The goals and outcome of a project to optimize TCP/IP transport over the FlexNet AX.25 network is described. A number of optimizations, and their implementations, are described and discussed. These include header compression, resend minimization, packet age tracking and ACK consolidation, as well as platform considerations and potential uses.

Not long ago, TCP/IP was an exotic mode of operation in Amateur Radio, reserved for specialists. Since then, it has become one of the most popular protocols used in the digital modes, due to the wealth of interesting applications employing it. The project described here attempts to make TCP/IP usage in the FlexNet packet network as simple as possible.

This doesn't mean that we can dispose of the de-facto AX.25 standard. No, TCP/IP is merely a useful expansion, or just another layer, that can be handled by FlexNet.

This also includes the reconnection of disconnected links without the intervention of the operator, including re-routing to faster links. TCP/IP can be seen as an end-to-end layer (Layer 4 in IS0 jargon), in contrast to other packet systems that are hop-to-hop layers, namely to the real ends of a logical circuit (user/user or user/server). TCP/IP strengthens the robustness of the AX.25 connection to the entry node, allowing one to change user ports during a running connection, without which the connection must be newly established. Instead of creating our own Layer 4 specification (plans have existed for some time), we find that TCP/IP offers exactly that, and is available for most modern operating systems. TCP/IP is the standard protocol of the Internet, and there is a considerable amount of software available.

TCP/IP and FlexNet make up a homogeneous unit, in that FlexNet nodes need no further expansions or changes, because they are already fully transparent to all frame types offered by the protocol, which are passed on a virtual AX.25 L2 connection. The protocols are passed along cleanly, as long as AX.25 (on the RF path) and TCP/IP (end-to-end) run at the layers they are expected to be at.

In our first look it was quickly seen that TCP/IP, as presently used in Amateur Radio, was given the (not undeserved) reputation of being slow and inefficient. This isn't the fault of the protocol, but of the implementations. It was clear where the crank needed to be turned. Quite simply, it shouldn't be like that, so we began this project. Today, a few thousand lines of source code that were implemented and tested on the first platform (Windows 95) are ready for use.

The idea of placing TCP/IP services directly on the network as AX.25 shouldn't be ignored. This is an interesting concept, but it should be pointed out that a careful TCP/IP implementation will return more than it costs.

Minimizing Protocol Overhead

TCP/IP packets contain a header of approx. 40 bytes. For a 256 byte packet, this is an overhead of some 16%, and, when the required ACKs are considered, more than 30% overhead increase as compared to an AX.25 connection. This also assumes an optimal protocol implementation, as well as no unnecessary retries. If you lengthen the packets to 1500 bytes (a typical value on Ethernet and similar implementations), the overhead sinks to a more reasonable 5%. A lengthening of AX.25 frames to 1500 bytes, as is often suggested, isn't practical. The frame failure rate would become unreasonably large: If an RF link of 256 byte frames is 90% failure free (a realistic (unfortunately) number), this value would decrease to only 40% error-free frames for 1500 byte packets. Sporadic interference such as radar impulses further worsen this value. For this reason, Ax.25 segmentation was defined some time ago, so that a large IP packet could be broken into many 256-byte packets. In this way, each frame contains only its own additional overhead bytes, contributing to efficiency. A requirement for this to function is that each packet arrives in the correct order, which can only be safely realized through a VC (Virtual Circuit) connection. The FlexNet AX.25 header compression feature *[implemented some time ago - IRZ]* functions in any case only with such a static connection and is a source of further reduced overhead.

These improvements are most efficient when there is a relatively large amount of data to be sent (enough to permit segmentation). If this isn't the case, such as in an interactive Telnet session, the value becomes somewhat less. The worst case is when each byte requires its own packet, with its 40 byte overhead. In these cases one needs some other mechanism. Luckily, there is a standardized header compression scheme, as seen in RFC 1144 [1], and implementation was relatively simple. Using the Van Jacobson scheme [2], the 40-byte header of a static TCP connection can be reduced to 3-8 bytes. the only assumption for this to work is, like the FlexNet header compression scheme, the connection path between both ends is static, i.e. a VC. This protocol was defined for relatively slow telephone connections. If you replace the concept of "Serial Line" with "AX.25 virtual circuit", you can see just how well the protocol would fit with AX.25 encapsulation. The compression and its control occurs on the AX.25 level for up to 250 TCP connections, as well as traffic forwarded through a router. With a Link Reset, the status tables on both sides of the path are re-initialized. This requires that both ends of the path know of these Link Resets, but not all AX.25 implementations do this, which can lead to problems in this area.

Thinking instead of Pumping

A general problem with end-to-end protocols is that the transport shell has only limited possibilities for influence. This is less important for a static TCP connection, where the timers regulate the connection fairly well on the basis of the transport capacity, but nonetheless, an orgy of unnecessary retries can occasionally be observed.

The goal of this project was to develop an IP router not only for usage under Windows 95, but for usage under DOS or on the RMNC platform. This gave us a reason to invest a little more work early on, to save work in the later cross-platform portings. Finally, we still have to co-exist with users and servers operating with network implementations that are sub-optimal.

With AX.25 these problems, once separated from channel control, are mostly resolved. The network nodes decouple both sides of a connection completely, and traditional digipeating is no longer practiced. With this philosophy, to an end-to-end approach such as IP becomes an attractive proposition.

IP is a connectionless protocol, and the TCP placed upon it is laid out such that packets can take any possible path from one end to the other, arriving in any order. For this reason, it is possible to route individual IP packets in a connectionless manner. An IP router might never see all of the packets of a particular connection. It is also true that when you encapsulate something within AX.25, it cannot be assumed that traffic from one side of a connection travels over the same path as traffic from the other side. Only at the endpoints can you be sure of seeing all packets of a TCP connection.

An IP router also has possibilities for optimization. If a router knows of congestion, for example the outgoing path in use is slower than the incoming path (not an unusual condition in a Packet network), it can analyze packets in transit stored in memory and eliminate all unnecessary resends, by simply erasing any doubled packets. A time stamp can also be put onto each packet. Packets that remain in the router for some time, say a minute (because of congestion, or a broken link) can be erased, hopefully resulting in a TCP retry from the sending station that might be routed along a better path. Implementation of this requires an analysis of the complete IP and TCP headers, so that the order of packets can be determined. This can also be used to consolidate ACKs for a specific TCP connection, passing only the latest to the endpoint.

Through such actions, congestion created by unnecessary resends can be dealt with instead of becoming worse. With traditional implementations, traffic is brought to a complete halt in such situations. Through controlled 'packet loss', as well as the deletion of doubled or aged packets, the net is much better able to deal with changes, slowing the data to match the path's capacity.

This behavior functions very effectively and completes the usual improvements, with the router sending an ICMP-source-quench as well as requesting the sender to throttle back. The retention of the originator is during this not clearly specified, thus such flow control actions can lead to very different results.

Naturally, both partners should adjust their TCP timers properly, but luckily this functions quite well, even with the Microsoft stacks.

The proper adjustment of TCPIIP parameters is critical, above all when one wants to reach destinations via different networks. The Microsoft stack is a "black box", and hardly any parameters can be adjusted. So, we need to make all optimizations in the module which knows the existing transport layer the best: The AX.25 coupler.

Instead of simply pumping TCP/IP packets into an AX.25 connection or, even worse, into AX.25 UI frames, as is done with present implementations (such as LinuX), the packets remain in the IP coupler's (e.g., router's) memory until they can actually be sent. This is comparable to the behavior of the FlexNet coupling to Layer 1, and contrasts with solutions such as KISS, which only generates frames that can be sent immediately. The known side effects of a KISS connection, i.e. multiple SABMs or RRs in a single transmit cycle, simply don't occur with FlexNet.

In FlexNet, ACKs are sent only for the latest packet, any ACKs belonging to earlier packets need not (and are not) sent. A carryover of this concept to the IP-AX.25 coupling brings some improvements. When the AX.25 connection is being established, during which no data can be passed, running TCP/IP packets are buffered, and so can cause unnecessary retires. The same happens with a user station that is blocked in a half-duplex connection for extended periods. During this time is it possible for many TCP packets to pass to the user station, each of which is ACK'd individually by the user's local TCP stack. Instead of sending each of these ACKs, the FlexNet coupler can discard all of them except for the last one. Implementing this requires a close coupling of the layers, but FlexNet already has the call in it's API, so that little accommodation is needed here. Naturally, the coupler must analyze and compare the packets at the TCP level, to be able to discard the proper packets. In the ideal case, a user station sends a maximum of one AX.25 I-frame during its time slot per active TCP connection, instead of the many L2 RRs and TCP ACKs as with the traditional solutions. The improvements thus realized are somewhat greater than those obtained from header reduction alone. While header reduction remains important, one cannot ignore the gains in efficiency from eliminating doubled or aged TCP packets.

In a typical HTTP session, many TCP connects are started, transferring only a few kilobytes of data before being closed again. TCP timers have no chance to adjust, and many unnecessary retires occur. This is especially true in a slow half-duplex environment which is typical for a user station. The situation is less dramatic when users have few, long-lasting connections, e.g. a single FTP transfer.

For AX.25, FlexNet was able to make such optimizations directly in the L2 code, because of the tight coupling to Ll . This avoids the need for the L1 process to have to analyze each L2 frame and decide what should be passed ahead or not. Unfortunately, it isn't so easy on the TCP level. Here, you must watch the status of possibly many running TCP connections, which requires a lot of memory and efficient algorithms. However, some coding efficiencies result because TCP compression needs quite similar data structures, which can be partially be used for these optimizations.

All this naturally increases demands upon memory space and CPU capacity. As long as the network runs at data rates below 100 kB/s, and PCs continue to increase in capacity, no further effort is needed. This is especially true on the user side, where nowadays there is more than enough computing capacity. A fast 486-class machine with 16 MB of RAM running Windows 95 is sufficient to support a throughput of 76,800 baud TCP/IP through FlexNet. For a router it would help to install a faster CPU, but reasonable performance with a 486 was measured.

Anachronisms

All the improvements described above assume IP transport over AX.25 Virtual Circuits, which should provide a reliable path between two points of an IP network. Using Datagram mode (AX.25 UI frames), each packet loss on the wireless portion of a TCP connection causes a TCP retry, which has to travel the whole way, end to end. (Remember the problems we had with L2 digipeating? Same applies here). Since this has not been generally learned and understood, an IP router must also be able to deal with the Datagram mode, where IP packets are sent as AX.25 UI frames. With this, one becomes stuck on the problem that only IP packets with a gross length of 256 bytes or less can be carried. AX.25 segmentation is not usable, as it depends upon the packets traversing the network in their original order.
To resolve this problem, we have available so-called IP fragmentation, which divides an IP packet into multiple smaller packets and gives (in contrast to AX.25 segmentation) each fragment it's own complete

IP header. Each packet is thus autonomous, and are reassembled in order only at the destination. This allows each packet to traverse the network through any available path, forbidding reassembly in routers or gateways. One problem is that when one fragment is lost, the entire packet (a series of fragments) must be re-sent. This scheme is clearly at best a temporary patch, especially when one must deal with the realities of the RF medium and the relatively high failure rate of the links. For compatibility reasons, Datagram mode is presently the only way to deliver a fragmented packet, and thus IP fragmentation must be implemented in the router.

IP Routing

IP routing is, in one regard, in competition with established AX.25 routing, especially in the FlexNet environment. In that IP addresses are mated to a callsign (dynamic address allocation for users is a subject for separate discussion), FlexNet routing is sufficient to reach a specified destination. One useful addition could be to define a layered hierarchical routing, such as AX.25 routing in the local area and IP routing for the wide area, analogous to protocol layering. An obstacle to this is that our network is not yet fully developed to offer fast connections with transit times of less than a minute over its entire range.

One argument in favor of IP routers at network facilities is that the user is freed from some work - he simply sends packets to the router, and it handles the rest. At the moment this merely means that the sysop's work, in which he might not have much interest, is delegated. The user shouldn't become dependent upon this system, however, he must retain the possibility of establishing a connection with a destination directly. This means that PID transparency, reversible callsign fields and a functional AX.25 router remain important. The network can be considered a cloud, its internal operation the user need not know to reach a destination.

With more and more users operating TCP/IP (perhaps in part due to this project), and TCP/IP is understood to be the end-to-end layer on a well-constructed AX.25 network, we can then reconsider the optimal bundling of traffic via dedicated routers. The infrastructure is, in the form of many LinuX systems, more and more available. All that was missing were the protocols and their implementation.

If the TCP traffic is optimized as discussed above, then routers can reduce network loading considerably. Routing, however, remains in the background. In any case, this is an exciting direction for development and there is considerable room for new ideas.

Until then, and possibly also after that, the user is better served with making a direct connection himself, as far as the network allows.

Which platform is the right one?

While we lean towards additions to some specific operating system, the rest of the world wants missionaries. Indisputably LinuX, for example, is an outstanding server/operating system. The average user, however, wants to simply buy something from Microsoft. You just can't beat Windows 95 or NT when you want to develop a user application.

For Windows 3.x, ETHEREMU from Thomas Sailer, HB9JNX, could be installed. It emulates an ethernet card and allows it to communicate via Trumpet Winsockets. The AX.25 routes are input using text mode commands.

TCP/IP is already integrated with Windows 95. Of course data transfer via Ethernet or "DFU-Network", using SLIP or PPP protocol, is provided. Until now, what was missing was a coupler that could encapsulate TCP/IP packets within AX.25. Some solutions already exist, but are generally difficult to configure, for example NOS in a DOS-box. Others try to use protocols such as SLIP, which has the disadvantage of removing the possibility of being able to have normal AX.25 connections. Besides, one needs special, and not quite inexpensive, hardware.

In the meantime, PC/FlexNet runs under Windows 95, all that's missing is the coupler that hands the IP packets over to FlexNet. Microsoft makes this possible only via the installation of a "Network Card", however the concept is, at least in the German translation, somewhat misleading. Of course, all of the optimizations described have been implemented, using Windows 95 as a test platform. Although the timing of the TCP stacks is really set up for a fast wire connection, it keeps stations on the RF channel fairly civilized and sends practically no unnecessary packets. A list box is used for AX.25 route inputs, and it also serves as a status and statistics display. No special TNC hardware is required, and running multiple IP sessions in parallel with AX.25 sessions presents no problems. However, this is a solution mainly for users, i.e. network clients.

As a server system, Windows 95 is somewhat strained. While there are some simple FTP and HTTP servers that function well, the entire system in not stable enough for use as a general-purpose server. One negative is that there is (still) no possibility of forwarding IP between multiple ports, as well as AX.25 to ethernet ports. This is the domain of LinuX, but NT is close behind. LinuX is already available with AX.25, so the next development point lies with NT. Having a large installed user base for 95 and NT doesn't hurt, either.

Uses

As already mentioned, applications and services are sufficiently available. Much that is developed for the Internet is also usable in Amateur Radio. Regardless, that shouldn't prevent someone from developing an amateur-specific program. Services such as databases or special applications like DX-Clusters are easier to access and service using TCP/IP instead of AX.25. Thanks to TCP port addressing, it's no problem to offer and try various applications and services on the same server with the same AX.25 callsign.

A further field for new applications is for image and speech transfer. While these demands today seem somewhat utopian, voice mailboxes are already using the network to transfer their messages via store & forward. It isn't a much larger step to make it possible for the user to have a direct digital connection into the system. The software already exists, though this is mostly designed for the relatively high speed of the Internet. If you don't need real-time and full duplex, and can deal with PTT (amateurs know this already) one can get very good results. Modern codecs allow speech to be compressed to less than 300 bytes/sec with little loss in quality. This bandwidth is already available in most of the network. In the future, a Windows application will be introduced. With specific servers it should also be possible to have roundtable discussions like on the local FM repeater.

Image transmission also remains in the range of possibility, especially with the quality codecs available for moving picture transmission. Naturally the existing available bandwidth won't work for decent quality video, but you don't really need it to admire someone's photograph. Convenient video conference systems are realizable in the foreseeable future.

All this carries with it the ability to increase the play value of the network, and through that, justify the spectrum being used. Naturally, our packet-oriented data transfer system isn't set up for real-time uses. The protocols and applications to be used require careful consideration of our unique requirements and conditions, but there is a lot of room for experimentation here.

Unquestionably, a careful optimization of all parts of the transfer chain is needed. What is lost in one component, whether in hardware or in software, cannot be recovered. For example, while increasing the baud rate is a good idea, it is not the only possibility for improving the network. Clearly, TCP/IP can bring to Amateur Radio much more than just a wireless Internet.

References:

[1] Jacobson, V., Network Working Group, Request for Comments 1144, February 1990.
[2] See also "TCP header compression according to Van Jacobson via AX.25" (Jost) elsewhere in these proceedings.