OSI Services on TCP/IP Networks

Anders Klemets, SM0RGV
Stephen Pink, KF1Y

Swedish Institute of Computer Science
Box 1263
S-164 28 Kista
Sweden

ABSTRACT

This paper describes the design and implementation of a method for providing upper-layer OSI services on top of a TCP/IP/AX.25 protocol stack. The tools for the project include the KA9Q Internet package in common use today in amateur packet radio, and ISODE, an ISO development package designed for wire-based networks but modified for packet radio use. The method used is described in DARPA/Internet RFC-1006 and is a standard for those in the Internet community who implement ISO protocols using TCP/IP based networks.

The paper first describes the network architecture of RFC-1006 and the basic scenario surrounding the implementation. A description of the implementation follows as well as plans for future development. The paper ends with a conclusion and a list of references to recent work.

Introduction

Packet radio networks have evolved from simple signaling systems to complex structures of communication using multi-layered protocol design. In many ways the evolution of packet radio communication has paralleled the evolution of data communication over wire and other land-based media. There have been periods of experimentation followed by periods of standardization during which more and more users of the network have contributed to its technical progress.

We are now in a rather special period. In parallel with the debate going on in our governmental, educational and commercial institutions on the merits of OSI versus TCP/IP, a similar debate has already started in our amateur ranks.

The point of this paper is not to argue for one or the other sides of this debate. Rather we wish to report on some results we have achieved using what we consider the best of OSI and TCP/IP in their current form as the basis for amateur packet radio.

We have done this according to RFC-1006, also described in [RoseCass], a DARPA/Internet document, describing a method for achieving upper-layer OSI services on TCP/IP networks using the KA9Q Internet package along with a popular and publicly available implementation of the ISO specified OSI upper layers, ISODE. The KA9Q software is well known to the amateur packet radio community, but the ISODE package needs some introduction. ISODE stands for "*ISO D*evelopment *E*nvironment." It provides the functionality for services at layers 5, 6 and 7 (session, presentation and application) of the OSI reference model while leaving the lower layers (transport and below) to native or pre-existing implementations on the host computer. For a detailed description of the OSI stack and the ISO protocols, see [Tanen]. On Ethernets and other wire-based networks, ISODE has been married successfully to various lower layer implementations, such as X.25 and foremost to TCP/IP. Thus ISODE has been used to foster a transition from DARPA to OSI applications and as a method for both worlds to co-exist. It was only a matter of software development to realize this marriage in amateur packet radio.

Most of the debate on the merits of OSI versus TCP/IP in the amateur community have centered on the lower or middle layers (network and transport.) It seems to us that the promise of OSI is in the upper layers, particularly in the richness of applications which have been recently defined and proposed by the ISO and other international bodies. Applications such as File Transfer Access and Management (FTAM), Message Handling Service (X.400) and Virtual Terminal *seem* to offer levels of functionality yet unavailable in the DARPA suite of applications (FTP, SMTP and Telnet) and *may*

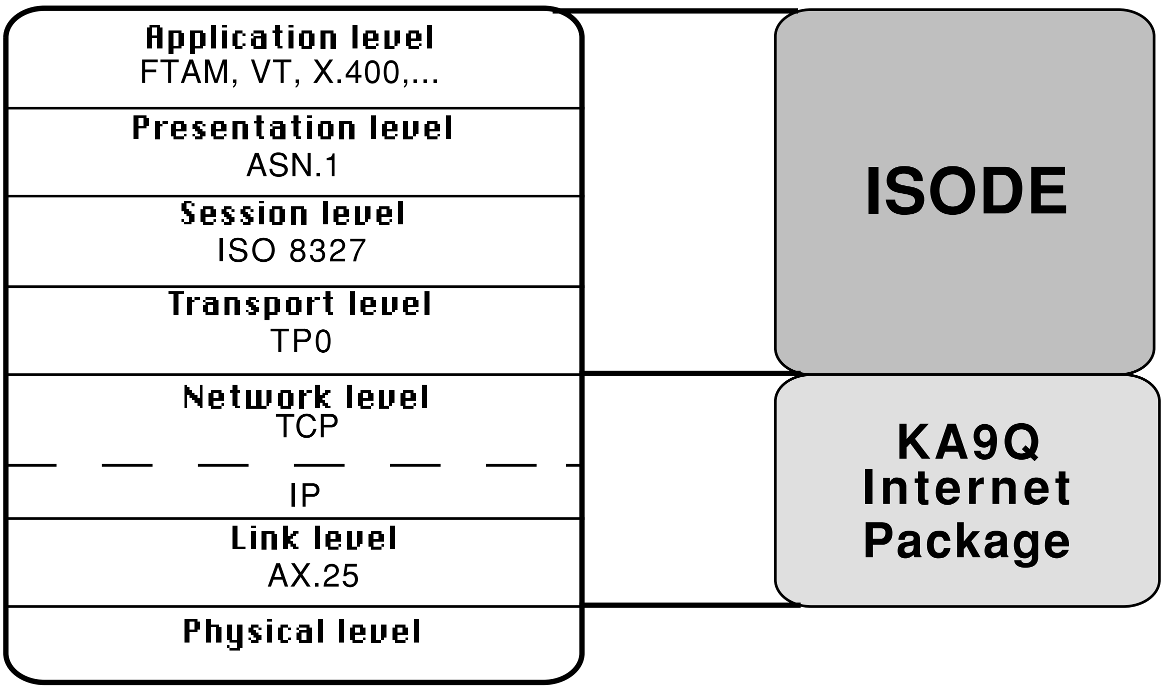

Figure 1: RFC-1006 as implemented with ISODE and the KA9Q Internet package.

offer the amateur community useful new services. In any case, we will never find out the truth of these claims for OSI unless we begin to experiment with these protocols.

The Scenario

According to RFC-1006, OSI services are achieved atop a TCP/IP network by establishing a Transport Service Access Point or TSAP upon which one then builds the session, presentation and application protocol agents prescribed by the ISO. The simplest way to do this, according to the RFC, is to encapsulate the least complicated ISO Transport Protocol, commonly known as TP0 (CCITT X.224 Class 0), inside TCP. The transport protocol TP0 requires the network service beneath to be reliable in an end-to-end sense. This means that the network guarantees that the packets on the receiving end are ordered in the same way as they were on the sending end before they are delivered to the network user, or in this case, the transport entity TP0. Fortunately, the combination of TCP and IP do exactly that. They provide a "virtual circuit" network upon which upper level services can be built. (See Figure 1.) Instead of thinking of TCP as analogous to the OSI Transport Level and IP to the OSI Network Level, RFC-1006 asks us to consider the combination of TCP and IP as providing a connection-oriented Network service similar to the service provided by the ISO protocol X.25.

Practically speaking, this scenario can be realized by using any implementation of TCP/IP with the proper programming interface to the ISODE software package which contains a TP0 implementation and a socket interface to TCP. Since the new KA9Q Internet software [Karn 88] (often referred to as NOS) provides TCP/IP and a socket interface for building new applications, the "construction job" was relatively easy. From the point of view of ISODE, the KA9Q software provides an OSI-like connection-oriented network service, and from the point of view of the KA9Q software, ISODE is just one more application using its socket interface.

We had to make a number of design decisions when we started this development; some of them were forced on us, others were made for the sake of modularity. First, we had to port NOS from MS-DOS to UNIX[1]. The size of ISODE in memory was so large, that any hope of running the ISO protocols from this package on MS-DOS was abandoned. This is of course a major disadvantage. Not every amateur has the luxury of a computer running UNIX in his shack. On the other hand, UNIX is starting to appear more frequently on personal computers and, with the popularity of the new 80386 machines increasing, it seems that UNIX may become a standard PC operating system in the future. UNIX provides a process address space large enough to hold the ISODE implementations of the ISO protocols we wished to use because it offers paged virtual memory. ISODE is written in C and meant to be run in UNIX environments, so it was a natural decision to change to UNIX despite the fact that UNIX is not yet popular among amateurs. It is perhaps a significant point about this particular implementation of ISO protocols that their size dictated a change in operating system environment. It seems characteristic of the ISO protocols in general that full implementations grow to rather huge sizes. Whether this size is necessary for any implementation is the subject of another debate.

Once done, the choice of UNIX as the operating system for our experimentation offered us some distinct advantages.

1. UNIX is a trademark of AT&T.

ISODE was already being run at our site over a number of different sub-networks such as the Swedish Telecom's X.25 Public Data Network and a TCP/IP Ethernet local area network with a gateway to the DARPA/Internet. Thus an AX.25 sub-network interface could be added in a modular way.

Design considerations

Although it would be possible to write a special driver for AX.25 that sits below the socket interface and the native TCP/IP code in a machine running Berkeley UNIX, this approach has some disadvantages. The main one being that computers running Berkeley UNIX are rather expensive.

There are a few radio amateurs that own UNIX computers, but these are often smaller models, eg. 80386 machines. They usually run AT&T UNIX System V, that does not have TCP/IP except as an expensive add-on.

An AX.25 driver for Berkeley UNIX has been written [Neuman], but there are some problems with that approach. For instance, the Berkeley TCP implementation expects shorter response times than what may be possible to achieve on a packet radio link.

The KA9Q Internet package [Karn 87], on the other hand, is written especially for use over packet radio. Its TCP implementation does not have timeouts built in. It also includes new mechanisms for congestion control. This makes the KA9Q TCP implementation robust in the face of long outages and the heavy congestion one may experience on packet radio channels.

Because the source code for the KA9Q Internet package is freely available we had another strong reason to choose it as the basis for this effort.

Porting NOS to UNIX

We already had the pre-NOS KA9Q Internet package running on a Sun workstation, but unfortunately it did not provide a network interface that we could use. It used a "commutator loop" to switch between different tasks. This is far from transparent to the applications.

However, the new version of the package, NOS, provided precisely what we needed. Its socket interface makes it much easier to write application programs for the various networking protocols supported by the KA9Q package.

As an initial effort, the NOS program was ported, with as few changes as possible, to different UNIX systems. This was fairly easy to do on a Sun-3 running SunOS 4.0. On our 80386 machine running UNIX System V release 3.2, however, things turned out to be more problematic.

There were three main problems. Firstly, a timer interrupt mechanism is needed. This can be implemented with UNIX signals. The timer interrupt decides what granularity one will get in retransmission timers, etc. Although getting a timer interrupt only once per second will work, one would like to have better resolution. On a Berkeley UNIX machine this can be done with the ualarm() system call. But UNIX System V alarm() provides a maximum granularity of one second. The solution to this was to let NOS create a child process that makes a poll() system call and then sends a signal to its parent. This system call can be made to sleep for times less than one second. Unfortunately, our System V machine occasionally fails to catch the signal as it should, and terminates the program. This is yet unresolved.

Secondly, the NOS program must get an

interrupt when one or more characters are available. On SunOS 4.0, ttys are implemented as STREAMS drivers. This makes it possible to get a signal every time new data arrives.

In the version of UNIX system V that we have this is not possible, since the serial line ttys are not implemented as STREAMS drivers. Instead, we had to resort to a loop that polls the ttys once every 10 milliseconds. Obviously, this causes a major performance degradation.

Finally, to achieve its multitasking, NOS assigns a separate stack to each of its processes. These stack areas should be in the data segment of NOS to make it possible to both read and write the stacks. This works on a MS-DOS machine and on a Sun-3, but on a Sun-4 and our 80386 machine running UNIX System V, one is not allowed to move the stack pointer into the data segment. The only solution we see is to place the stacks in some spare area below the original position of the stack pointer, but above the heap.

Link level interfacing

We did some initial tests with ISODE over packet radio without modifying the ISODE source code. The FTAM client transmitted its IP datagrams on an Ethernet. The Ethernet packets were intercepted by NOS. It routed the IP datagrams onto the radio channel to another NOS program. This NOS program routed the datagrams to the FTAM server over an Ethernet.

Although this method works, it has some disadvantages. The end point machines are completely unaware that their IP datagrams are routed through a packet radio channel. Therefore the datagrams have sizes suitable for Ethernets, approximately 1500 bytes. If these IP datagrams are sent in 256 byte AX.25 frames, they have to be segmented, which is something that should be avoided. Also, the TCP implementation will timeout if the throughput worsens.

Network level interfacing

The problems described above made it clear to us that it was inefficient to use the KA9Q software as a IP router only. It must be used by the application programs in an end to end sense. ISODE should use at least KA9Q TCP/IP as its reliable network service.

There are several ways of implementing this. One could make the services provided by ISODE (FTAM, VT, etc.) a part of the NOS program. But the program would become so big that it would not be possible to run it on an MS-DOS machine.

Furthermore, the multitasking in NOS is done non-preemptively. This means that every process runs without interruption until it needs to wait. This works very well with communication protocols, since they often need to wait. But there might be high level applications that do not wait as often. Typical examples of this are programs that search through large databases on disk or tape. It is not easy to make these and similar applications work well when the operating system scheduler is non-preemptive.

The most obvious objection against making ISODE a part of the NOS program is that UNIX is already multitasking. Having a program that implements its own multitasking adds unnecessary overhead.

Separate UNIX processes

We decided to eliminate the multitasking in NOS by splitting it up into several UNIX processes. Our implementation, whose working name is NETNIX, is shown in figure 2.

All transport, network and link protocols in NOS are run in a background process on the UNIX machine. Another process handles input from the tty to which the TNC is attached. This process has only one tty to serve, so it can block indefinitely, instead of polling it every 10 milliseconds. This eliminates one of the major problems with running the KA9Q software on System V machines.

The application programs, such as FTP, etc., do indeed run as separate programs under UNIX. They communicate with the server program using normal UNIX System V shared memory and semaphores. This interprocess communication can be made invisible to the application programmer. All he needs to know is how to use the socket interface, which he links into the program at compile time. This socket interface is similar to the one provided by Berkeley UNIX systems, but is implemented using the socket code in NOS.

All application programs and the server have access to a pool of shared memory. They allocate buffers (mbufs) from this pool and transfer the data by passing pointers to the appropriate buffers. The socket table and a few global variables are also kept in shared memory.

When an application program makes a call to a socket interface function, such as socket() or connect(), it is executed in its own process space rather than in the server process. Eventually there is a need to call some protocol specific function, like open_tcp(). The application program then issues a remote procedure call to the server, telling it to execute the specified function with the specified parameters. When the server has finished executing the function, it returns a value to the application program. The remote procedure calls are implemented by passing pointers to buffers in shared memory.

The main disadvantage with this imple-

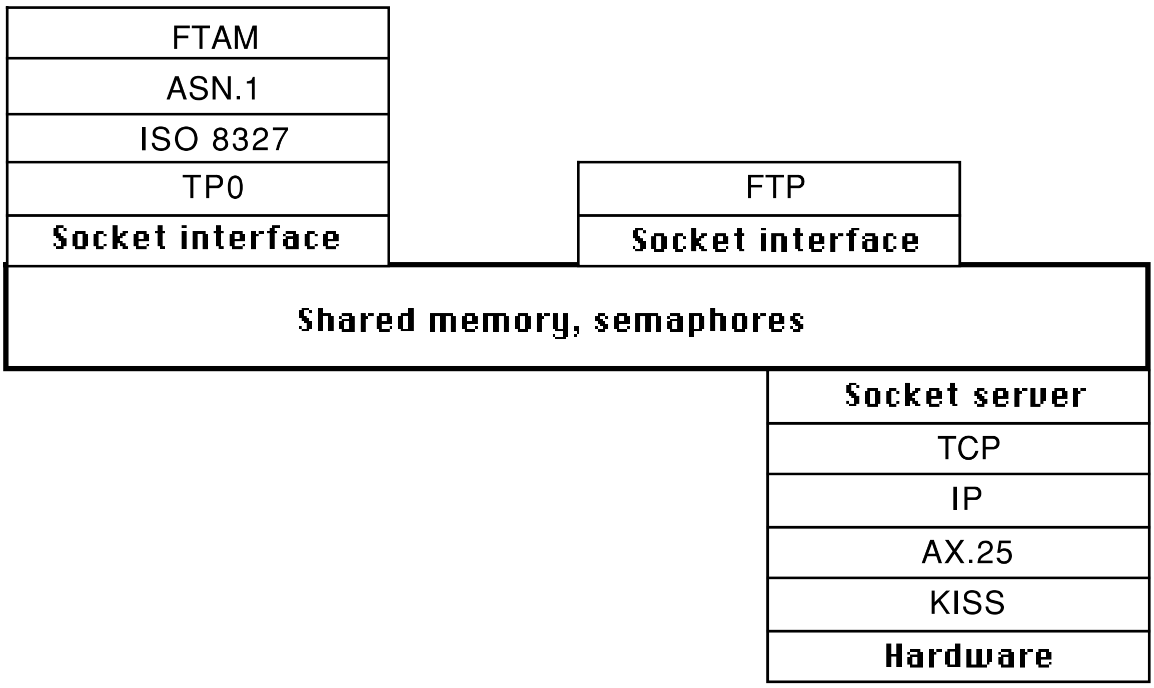

Figure 2: Two application programs communicating using the NETNIX shared memory, semaphore and socket interfaces.

mentation is that all critical sections have to be isolated with semaphores. Otherwise strange things might happen, such as two processes trying to write to the same piece of memory at the same time. This extra overhead, and the fact that the protocols run in user space, forcing context switches, make the implementation somewhat slower compared to native TCP/IP code on a Sun workstation, but it is still acceptable.

Transport level interfacing

ISODE has a modular interface between the session level protocol and the transport service. So, it would be possible to incorporate a TP0 transport protocol into NOS or NETNIX.

But unlike TCP, there is nothing to gain by trying to customize TP0 for use over AX.25 packet radio links. The main purpose of TP0 is to provide a transport service access point, it does very little else as a protocol. Thus, we can use the TP0 implementation just as it is included in ISODE. There is no pressing need to build our own OSI-conformant transport protocol on top of TCP/IP.

Implementing protocols in the UNIX kernel

The protocol handling in NETNIX is fully implemented in user space. In operating systems like UNIX one usually puts the communication protocols, at least up to the transport level, into the operating system kernel. If the implementation is done properly there is no need to use semaphores to prevent conflicts between different users.

SunOS 4.0 and System V release 3 provide a mechanism called STREAMS for adding functionality to a communications channel in a modular way. Once a STREAMS device driver is opened, it is possible to push modules on top of it. All data that is written to the device may pass through this module that can add or

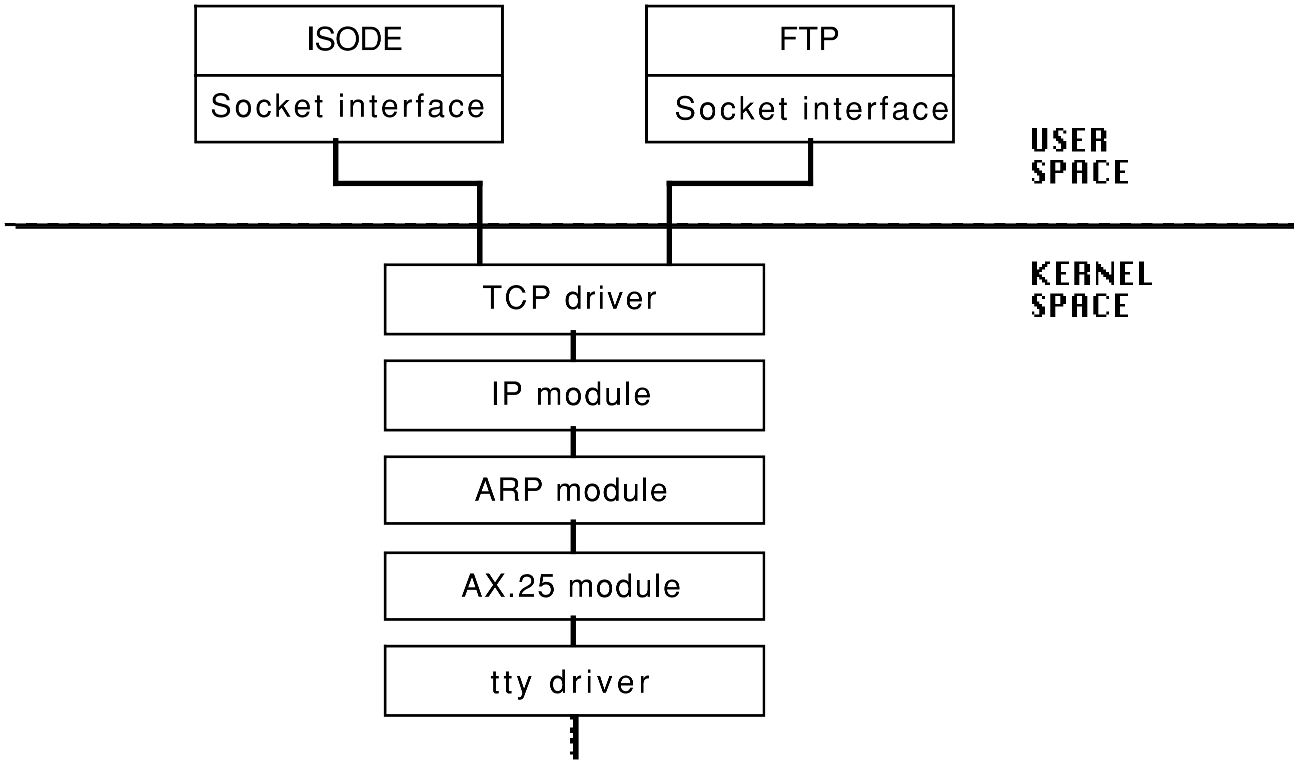

Figure 3: TCP/IP and RFC-1006 implemented with STREAMS.

remove protocol headers. It is also possible to send messages that are interpreted as specific commands to the modules. One may also build one´s own STREAMS drivers, which can multiplex in both directions.

At the time of the writing this paper, we have finished an AX.25 STREAMS module. This module can be pushed on top of a STREAMS tty driver. The AX.25 module itself does not know how to write characters to a serial port. This is handled by the underlying tty driver. An ARP module and an IP module are also near completion. Eventually a TCP driver will be written. It will sit on top of all other modules and allow several user processes to access it by means of UNIX file descriptors. (See figure 3.)

A Berkeley UNIX-like socket interface could be built as a set of library routines. These library routines would map the familiar calls such as bind(), connect(), etc., to a series of commands to the TCP driver in the kernel.

Conclusion

We have described an experiment achieving certain OSI services such as FTAM over TCP/IP/AX.25 packet radio links. We found that we could do this by modifying existing publicly available software.

Thus, we have created a testbed where further experimentation with ISO protocols on packet radio networks can be further researched and developed. We observed, among other things, that there was more processing overhead for the ISO protocols than for the DARPA application protocols and we need to eliminate this processing bottleneck before we recommend the everyday use of the ISODE part of our software. We do, however, believe that we have made the first step towards developing a usable OSI-conformant packet radio network[2].

Acknowledgements

We wish to thank Hakim Laraqui of Swedish Telecom Radio, who helped design and implement the ISODE packet radio interface. We owe debts of gratitude to Marshall Rose for writing ISODE, and to Phil Karn for writing the KA9Q Internet package.

References

Karn 87 Phil Karn, "The KA9Q Internet (TCP/IP) Package: A Progress Report", *ARRL 6th Computer Networking Conference*, August 1987, pp. 90-94.

Karn 88 Phil Karn, "Amateur TCP/IP: An Update", *ARRL 7th Computer Networking Conference*, October 1988, pp. 115-121.

Neuman Clifford Neuman, "Packet Radio and IP for the UNIX Operating System", *ARRL 6th Computer Networking* Conference, August 1987, pp. 143-147.

RoseCass M.T. Rose, D.E. Cass, "OSI Transport Services on Top of the TCP", *Computer Networks and ISDN Systems*, North-Holland, Vol. 12, No. 3, 1986, pp. 159-179.

Tanen Andrew S. Tanenbaum, "*Computer Networks, Second Edition*", Prentice-Hall, 1988.

2. The authors may be reached on the Internet as klemets@sics.se and steve@sics.se respectively.