Low Tech Connections into the ARPA Internet: The RawPacket Split-Gateway

Marshall T. Rose
Department of Information and Computer Science
University of California, Irvine
Fri Feb 17 23:06:24 1984

Computer Mail: MRose@UCI

Technical Report Number 216

ABSTRACT

This report describes a "low technology" method for connecting into the ARPA Internet. The use of a *RawPacket* interface in a system which supports IP makes possible the construction of a *split-gateway* between two hosts. The *RawPacket* interface permits a user-level process to introduce arbitrary packets into the IP layer, resulting in a virtual network interface. Since the *split-gateway* is implemented using a *RawPacket* interface, two networks may be connected together using a convenient medium which does not require explicit kernel support. Hence, *split-gateways* are well-suited for use as *stub-gateways*, connecting a local network to a long-haul network such as the ARPA backbone. In particular, the *split-gateway* discussed in this report achieves a reasonable level of connectivity for a comparatively small expenditure.

This report details how the *RawPacket* software and *split-gateways* are implemented. In addition, various daemon configurations are presented, modifications to the operating environment are discussed, and some performance measurements are given.

Table of Contents

Background

In the interests of brevity, we dispense with a lengthy, repetitive discussion of the usual networking concepts. Instead, we note that the topics discussed in this report are greatly intertwined with the behavior of a networking implementation that supports the Internet Protocol[IP]. As a result, familiarity on the part of the reader with the general notions of protocols and layering[OSI] is assumed. In addition, the reader should have an understanding of the concept of a "catenet," as described in [CERF78].

RawPacket Software

The *RawPacket* software discussed runs on 4.1aBSD VAX[1]/UNIX[2]systems, and, more recently, on 4.2BSD systems as well. The software consists of two parts, a *network interface*, which is kernel-level code, and a *daemon*, which is user-level code. The following discussion is more meaningful if the reader is somewhat familiar with the internals of the networking portions of the 4.2BSD kernel, and the system interface that the 4.2BSD kernel provides, so it might be useful to consult [LEFF83] and [JOY83] before continuing. In particular, a number of the concepts presented in [LEFF83] appear throughout this section.

As a brief reminder, 4.2BSD UNIX supports a number of protocol families. A protocol family consists of an addressing format, and a number of protocols, each of which implement a particular type of UNIX socket. Currently, three sockets abstractions are defined: **SOCK_STREAM**, which supports sequenced, reliable, two-way connection based byte streams; **SOCK_DGRAM**, which supports unordered, unreliable, connection-less datagram delivery; and, **SOCK_RAW**, which permits access to internet network interfaces. Our interest is with the Internet Protocol suite, and in particular IP. Hence, we are more likely to encounter those parts of the kernel dealing with the **SOCK_RAW** socket abstraction and the **AF_INET** address format, than those parts dealing with other types of sockets or protocol families.

RawPacket Interface

The *RawPacket* interface appears exactly like an ordinary network interface to the UNIX kernel. Unlike an interface to a hardware device, such as an ethernet or an IMP, the *RawPacket* interface does not have an associated hardware component. Instead, like the loopback interface, the network associated with the *RawPacket* interface is implemented in software.

When IP goes to send a packet to the network connected to the *RawPacket* device, the output routine for the *RawPacket* interface is invoked, given the packet to be output, along with a destination for the packet. This routine calls the standard input routine for raw sockets, claiming that the packet came from the network associated with the *RawPacket* interface, that the protocol family associated with the packet is the *RawPacket* family, that the address format associated with the packet is the *RawPacket* format, and that the packet is destined for the address it was given. This raw input routine places the packet into a queue for raw sockets, and then examines the raw sockets in the system to see if the packet can be given to one of them.

In contrast, the software loopback interface, when called to output a packet, simply places the packet on the IP input queue, as if it had read the packet from a device. This differs from the *RawPacket* interface in a subtle way: Instead of giving the packet back to IP, claiming that it had newly arrived from an interface, and letting the IP input routine

---

[1] VAX is a trademark of Digital Equipment Corporation.

[2] UNIX is a trademark of Bell Laboratories.

handle it, the *RawPacket* interface takes a packet from IP and gives it to a *RawPacket* socket for processing.

The algorithm used by kernel to determine which raw socket is given the datagram is relatively straightforward: the protocol families for the socket and the packet must agree, and if the socket was created for a particular protocol in the protocol family, then it too must agree. Further, if a local address is bound to the socket, then it must agree exactly with the destination. Similarly, if a foreign address is bound to the socket, then it and the packet's source address must agree. If no sockets exist which satisfy this criterion, then the packet is dropped. For our purposes, the kernel looks for a raw socket associated with the *RawPacket* protocol family, but without a specific protocol number in the *RawPacket* family. To simplify things a bit, the local address of the socket is whatever network is associated with the *RawPacket* device, and the socket does not have a binding for its foreign address.

When a socket using the *RawPacket* family sends a datagram, another routine in the *RawPacket* interface is invoked. This routine mimics the output routine for the loopback interface: it simply places the packet on the IP input queue. Hence, as far as IP is concerned, a network interface just read a packet. The entire contents of the packet is uninterpreted. As expected, this is the inverse of the operation described above: the *RawPacket* interface takes a packet from a *RawPacket* socket, and gives it to IP for processing.

RawPacket Daemon

In order for *RawPacket* to be useful, a daemon is required to open an *RawPacket* socket and accept datagrams given to it by the raw input routine. The daemon currently in use is called *rpd*. The function of the daemon is simple: any packets it reads from the raw socket it outputs somewhere else, and anything it gets back from that destination it writes back to the socket. It is easy to visualise two such daemons, on different hosts, connected via some medium exchanging packets. This forms the basis for a *split-gateway*, which is discussed later. It is important to understand that ***any*** medium could be used to connect the two daemons. Although *rpd* uses tty lines, it could just as possibly use punched cards[3].

Let us proceed to the point where the two daemons have somehow connected to each other over a terminal line, and consider what each daemon does. Each daemon opens the line in raw mode to avoid any processing of characters that might occur. In addition, parity checking is disabled along with echoing. This results (hopefully) in an unintepreted 8–bit data path between the two processes. Each daemon now opens a *RawPacket* socket on their respective hosts, and enters an endless loop.

[3] Assuming, of course, that UNIX supported such an outlandish device.

If something can be read on the *RawPacket* socket, *rpd* reads it, and sends the packet to its peer. To accomplish this, the daemon breaks the packet into a collection of 8–bit bytes, and sends this stream over the terminal line:

DLE STX ENCODED PACKET DLE ETX

where ENCODED PACKET is the contents of the packet with every DLE byte preceeding (a.k.a. escaped) by another DLE. The choice of values for the three characters is arbitrary, providing that both *rpd* peers agree to their values.

If something can be read on the tty line, *rpd* reads it into a byte queue. Starting at the beginning of the queue, the daemon looks for a DLE STX pair. Upon finding that, *rpd* copies the following characters to another buffer. If the daemon finds a DLE ETX pair, it declares the packet complete and writes it to the *RawPacket* socket. If *rpd* finds a DLE DLE pair, it copies only one DLE to the buffer to be written to the socket. The daemon handles corrupted packets by dropping them. For instance, if when reading a packet, *rpd* finds DLE X, where X isn't one of the three magic characters, it declares the current packet bogus, and starts looking for another DLE STX pair to start the next packet.

Implementation wise, the daemon does all of it's I/O in non-blocking mode, performing re-tries when appropriate. Further, *rpd* forks into two processes when it starts, one reads from the terminal, the other reads from the socket.

Split-Gateways

A *split-gateway* connects two networks together, like an "ordinary" gateway. Unlike the normal definition, in which a gateway is viewed as one host that resides on two (or more) networks, a *split-gateway* is thought of as being composed of two hosts (on separate networks) that are connected together by an internal communication path (which is, in fact, a third network). Each of the two "primary" networks see *only* their half of the *split-gateway*, and do not suspect that the *split-gateway* is actually two machines connected via another medium.

How the *Split-Gateway* Works

At present, the hosts **UCI-750a** and **ISI-Troll** form such a *split-gateway*, connecting the **UCIICS** network into the Internet. To put things simply, the ARPA backbone "knows" to give packets for **UCIICS** hosts to **Troll**, a host on **ISI-NET**, which in turn "knows" to give packets to the *RawPacket* interface, these packets are then given to the *RawPacket* daemon running on **ISI-Troll**, which transmits them to the *RawPacket* daemon running on **UCI-750a** over a 9600–baud leased line from TelCo, which gives them to the *RawPacket* interface. How the ARPA backbone knows to direct datagrams to **Troll** is not germane to our concerns; the remainder of the transit is explained here.

There is a daemon known as **routed** which manages the kernel's routing tables. Every 30 seconds or so, **routed** broadcasts a datagram to its peers on each network interface. The datagram summarizes the host's connectivity in terms of destinations and the "best" hops to those destinations. When it starts, **routed** reads **/etc/gateways** and adds the routing entries it finds there to the kernel's routing tables. Hence, an entry is made in this file declaring the network located at the other end of the *split-gateway* to be reachable via the address bound to the local *RawPacket* daemon.

For the file on **ISI-Troll**, this entry is:

dst uciics gateway rp-gw metric 1 passive

while the entry in **/etc/gateways** on **UCI-750a** is:

dst isi-net gateway rp-gw metric 1 passive

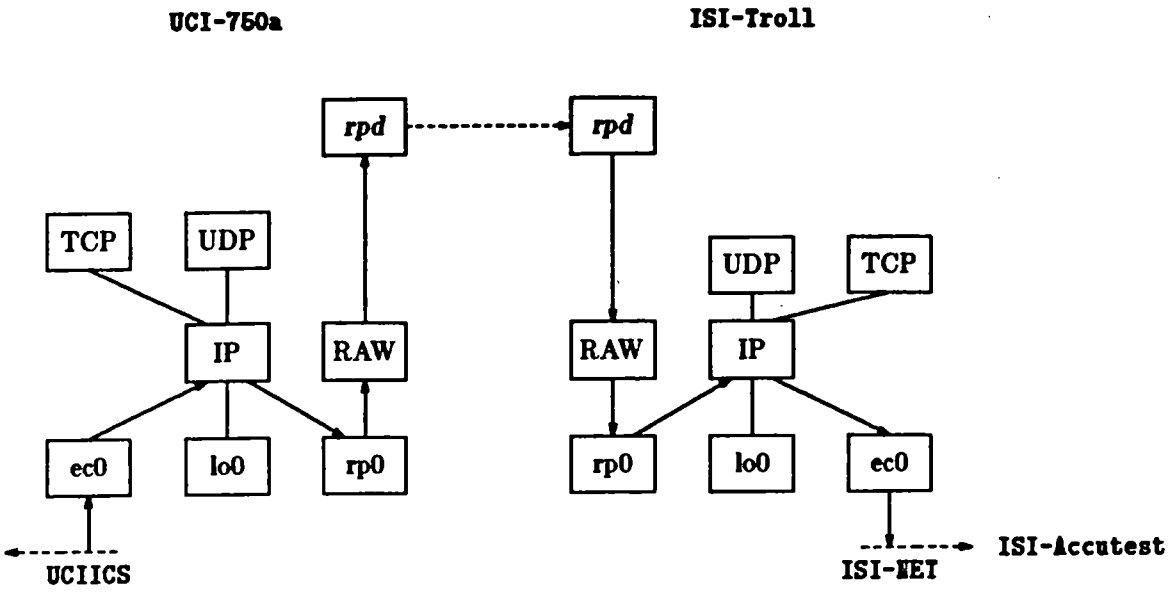


Figure 1. Path from UCIICS to the Internet

Both of these hosts are VAX-11/750 systems running 4.1aBSD UNIX. This version of the kernel does not support a "default" routing entry. Hence, each network that **UCIICS** wants to be able to reach must have an entry in **/etc/gateways**. For **Troll**, the entry directs traffic to the **ISI-Accutest** system which acts as a gateway between **ISI-NET** and the ARPAnet, while the entry on **UCI-750a** directs traffic to the *RawPacket* daemon. For **UCIICS**, **UCI-750a** functions as a stub-gateway to the Internet. Hence, a default routing entry would be very useful: if a route is not known to a particular destination, give datagrams for that destination to the *RawPacket* interface. Since this is not supported in 4.1aBSD, each network that a host in **UCIICS** wants to connect to must be entered into the **/etc/gateways** file on both **ISI-Troll** and **UCI-750a**. Intervention of this sort is annoying at best. When these hosts convert to 4.2BSD, a **default** gateway can be used to avoid these problems.

Figure 1 shows the path that a packet travels from the **UCIICS** network to the Internet. A packet is received on the ethernet interface (ec0:) by **UCI-750a**, which gives it to IP. IP consults the routing entry for the packet's destination and gives it to the *RawPacket* interface (rp0:). The *RawPacket* interface writes the packet to *rpd*, which in turns sends it across the leased line, to its peer on **ISI-Troll**. The peer writes the packet to the *RawPacket* interface, which gives it to IP. IP consults the routing entry for the packet's destination and gives it to the ethernet interface for **ISI-Accutest**, which enters the packet into the ARPA subnet.

A Note on Numbers

In our context, a network interface has two addresses associated with it: a ***network address***, which is the number of the network that the interface is connected to; and, a ***host address***, which is the address of the local host on that network. The *RawPacket*

interface is no exception, the number chosen for the interface's network address is network 126. As the reader has probably guessed, the name **rp-gw**, is used to denote this number. Although it makes sense to assign individual numbers to the two hosts composing the *split-gateway*, it makes no operational difference, so **rp-gw** is used to represent the host number as well.

It should be noted that net 126 is not an approved or official designation for **rp-gw**[1]. Fortunately, this network is known only to the two hosts forming the *split-gateway*, and problems do not manifest themselves.

Multiple *Split-Gateways*

Given the previous explanation of the way the UNIX kernel selects a *RawPacket* socket to give a datagram to, it should be a straightforward exercise to envision how a host could participate in more than one *split-gateway*[2]. That is, it should be possible for a host to act as half of several instances of a *split-gateway* at the same time.

The solution, of course, is to run a *RawPacket* daemon for each *split-gateway*, and have each daemon open a *RawPacket* socket using a different binding for the local address. The routing tables for this host are then modified to reflect each of these addresses as servicing the appropriate destination.

Known Problems

At present, there is one outstanding problem which has not been satisfactorily dealt with. The largest datagram that the *RawPacket* interface will accept is 512 bytes (in UNIX parlance, the **MTU** of the *RawPacket* interface is 512). If a protocol such as TCP sends a large segment, IP must perform fragmentation to break the resulting datagram into chunks small enough to be acceptable to the *RawPacket* interface.

For reasons not yet determined, these fragments are not correctly re-assembled at the other end of the *split-gateway*. Since both halves of the *split-gateway* are running 4.1aBSD, a bug in 4.1a's fragmentation or re-assembly algorithm is suspected. Ironically, the only hosts on the Internet that experienced troubles in communication with hosts on **UCIICS** are sites running 4.2BSD UNIX, owing to the large segment sizes selected by the 4.2BSD TCP implementation. As of this writing, this problem is still unresolved, although there are a few leads on it[3].

---

[1] The software loopback interface, which uses net 127, and host 127.1, is similarly guilty of this transgression.

[2] Truth in advertising requires that the following statement be made: Although the author has every confidence that multiple *split-gateways* will work as described, sadly, the opportunity to verify their behavior has not yet arisen.

[3] This problem may well become moot when both hosts composing the *split-gateway* convert to 4.2BSD which should not have these difficulties.

Running *RawPacket*

There's a bit more to running *RawPacket* that just setting up a couple of daemons connected via a tty line. We now consider some of the different ways that *rpd* can be configured to run between two hosts. Following that, we examine a few extra modifications made to UNIX to support *RawPacket* more efficiently. Finally, we scrutinize the bottom line by seeing how well *RawPacket* performs.

RawPacket Daemon Configurations

Using some method, the two daemons must connect with each other before the *split-gateway* can operate. There are really two ways to set this up: establish a "dedicated" tty line between the two, or have one daemon "call" the other one.

In the first case, it is necessary to tell `init` on each host that the tty line is not for general use. This is easily done by modifying `/etc/ttys`. Next, each host modifies its `/etc/rc.local` file to invoke *rpd* using the correct tty at system startup. This method has the advantage that either host can go down or come up arbitrarily and no operator interaction is required.

In the second case, one of the hosts requires a dialer (or similar device) connected to a tty line, so it can establish a connection to the other host composing the *split-gateway*. To facilitate this, *rpd* has a built-in script interpreter that reads a series of commands describing how to dial other systems and what responses to expect. The host doing the calling is known as the *active* host, and the `/etc/rc.local` file on the active host is modified to invoke *rpd* with the proper script name at system startup.

The host being called is known as the *passive* host. The administrator on the passive host establishes a UNIX login for the active host. Typically the active host's script, after logging a job in on the passive system, runs the *rpd* program. Although this configuration can restart if the active host goes up or down, some sort of intervention is usually required if the passive host goes down or comes up. In order to avoid manual intervention, not only does *rpd* have to constantly retry if it can't establish a connection to the passive host, but it also has to know (somehow) that the passive host has indeed gone down.

For production mode service, the direct-connect configuration is desirable, while the calling configuration is good for debugging purposes.

More Modifications to UNIX

In order to get a reasonable data transfer rate, with all of the protocols being layered over the *RawPacket* medium, a 9600–baud link is essential. Unfortunately, UNIX is typically unable to handle a burst of traffic at high speeds on a tty line using either the new or old tty drivers. At first glance, a good solution is to use the *BerkNet* (or "network") line driver, since it was specifically designed to handle large amounts of continuous input over a terminal line. The *Berknet* line discipline buffers up to 512 characters from the tty line without any processing whatsoever. A record is terminated in the buffer by a newline character (which is what we conveniently use as the ETX character). If a process tries to

read from a tty using this discipline, it blocks until the newline is received and the record is terminated. Any characters arriving on the tty line after a record has been terminated, but before it has been read, are discarded. The very limited processing performed by UNIX permits a process to read large amounts of information efficiently from a tty line with little system overhead. It turns out when using the new tty driver, even in raw mode, UNIX executes a lot of code for each character input, by testing various conditions, and usually ends up inserting the character into a queue. In contrast, the network line driver executes a few (typically 3) lines of code for each character received.

A glaring deficiency of the *Berknet* line discipline is that it supports a 7–bit path only. So, the first modification to be made is to permit 8–bit bytes to be transferred. This change to UNIX is trivial, providing that any other applications making use of the network line driver are prepared for this behavior.

A second problem, is that the *RawPacket* daemon uses some of the new BSD UNIX I/O facilities, such as the **select** system call to perform synchronous I/O multiplexing. For reasons unknown, the 4.2BSD kernel did not implement the **select** facility for terminals using the network line discipline. A line of code here, a line of code there, and it all works as expected.

Finally, if the system is loaded, it is possible that some characters could be dropped if *rpd* did not wake-up soon enough to empty the kernel's buffer before the next packet arrived. To protect against this happening too often, the *Berknet* line driver was modified once again. This new modification allows it to continue to stuff characters into the buffer even if they arrive after the record was terminated but before the daemon could empty it. Obviously, no characters are accepted once the buffer is full, so they are dropped.

A Few Other Changes

If more than one 4.2BSD host is on the same local network that supports a *broadcast* mechanism, then the **rwho** and **ruptime** commands provide an interesting facility. Invoking **rwho** lists the users on each host, while running **ruptime** lists a summary of the number of users and the load average on each host, along with the amount of time since each host was booted. Clearly, it would also be useful to know how long the *split-gateway*, "the link to the outside world," has been running. There are different ways that this can be accomplished, the one implemented is described briefly here.

As previously mentioned, **routed** manages the routing tables for the kernel, by sending a datagram to one or more peers on each network interface every 30 seconds or so. Since the *RawPacket* interface appears as any other network interface to **routed**, it sends a datagram to its peer at the other end of the *split-gateway*. As a result, in a normally operating *split-gateway*, some traffic every 30 seconds should be expected in both directions. All **routed** need do is note when it receives a datagram from its peer at the other end of the *split-gateway*, and tell the daemon that keeps track of the information used by the **rwho** and **ruptime** programs, **rwhod**. This is currently implemented by having **routed** send a broadcast datagram on the local network, in the standard **rwhod** datagram

Load	Disk	Bytes	Seconds	Baud
7.29	24	175067	253	5528
7.92	40	66256	109	4856
3.79	28	175067	241	5808
3.04	10	66256	82	6464
2.32	6	175067	234	5984
2.20	15	66256	84	6304

Figure 2. Informal Performance Measures

format, with the appropriate information (e.g., host "internet" has been up for 5 hours and 17 minutes[1]). Further, **rwhod** was modified to accept packets from a **routed** process.

Although not the most elegant solution, it gets the job done and is very reliable. It turns out that being able to invoke **ruptime** and find out how long the *split-gateway* has been up is very useful for a number of reasons. It should be noted that the modifications to **routed** and **rwhod** (along with a cosmetic change to **ruptime**) were part of a "big hack attack."

Finally, the **rstat** program, which reports on the status of the network from the local host's perspective required some modification: the 4.1aBSD version had some bugs when displaying information for raw sockets, and the 4.2BSD version didn't handle raw sockets at all. Both of these problems got fixed.

Performance Measurements

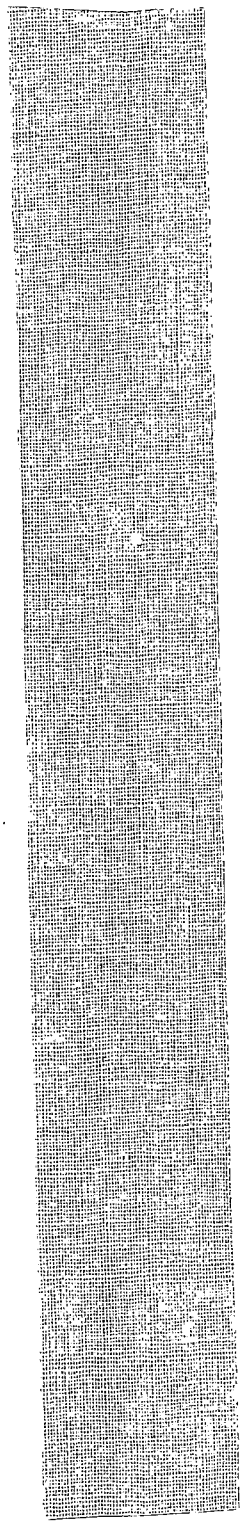
Figure 2 presents some informal measurements that were made on FTP transfers between **UCI-750a** and **SRI-NIC**. The *split-gateway* was operated over a 9600–baud line between **Troll** and **UCI-750a**.

The **Load** column identifies the "5 minute load average" of processes waiting to run on **UCI-750a** when the transfer took place. Similarly, the **Disk** column identifies the "disk load average[2]." **ISI-Troll** is a lightly used system, with a 5 minute load average varying from 0.50 to 1.25, and a disk average of about 5 to 10. Although these averages are not infallible metrics of the system's load, they do provide some insight into how busy the **UCI-750a** was when these transfers took place.

It appears that with a moderate 5 minute load average of 3 or so, a data transfer rate of 6000–baud can be achieved. This rate shows the speed that bytes can travel from one user-level process to another (e.g., FTP client and server), and can be considered to be the effective end-to-end throughput provided by the *split-gateway*.

[1] The uptime can be calculated by keeping track of when **routed** started and how long ago the previous datagram was received.

[2] This number is calculated by examining the values of **_dk_xfer** and **_cp_time** in the kernel and performing some arcane calculations.



The costs of establishing the *split-gateway* was inexpensive, though not insignificant. The 9600–baud leased line from TelCo cost $750.00 to install with a monthly charge of $250.00. The modems cost $2560.00 each, and a pair of asynchronous/synchronous interfaces cost $335.00 each. Hence, for a start-up cost of about $6300.00, a monthly charge of $250.00, and some goodwill from the people at ISI, **UCIICS** was able to connect into the Internet.

Closing Statement

The *RawPacket* interface supporting a *split-gateway* is an inexpensive method of establishing a stub-gateway for a local network. This provides the hosts on the local network with connectivity with the Internet, by forming a *split-gateway* with a "sponsor" Internet host. Since the *RawPacket* interface relies on a user-level process to do actual transmission of packets between the ends of the *split-gateway*, the use of a convenient medium can be exploited. Further, this makes possible experimentation with different media and paths, without requiring kernel modifications or downtime once the *RawPacket* interface is installed. Although user-level code partly implements the system, measurements of end-to-end throughput show that "reasonable" data transfer rates can be achieved.

Acknowledgements

The code implementing the *RawPacket* interface in the 4.1aBSD and 4.1cBSD UNIX kernels was originally developed by David A. Kashtan at SRI International. In addition, Mr. Kashtan supplied a VMS[1]/EUNICE version of the *RawPacket* daemon. Without his efforts, the *split-gateway* between **UCI-750a** and **Troll** would not have come about.

It should be noted that independent the of development of the *RawPacket* implementation described in this report, Christopher A. Kent of Purdue-CS introduced a similar capability for systems running the BBN TCP/IP implementation[GURW81] for 4.1BSD VAX/UNIX.

The script interpreter found in *rpd* was inspired by a similar facility present in the PhoneNet channel found in the University of Delaware's Multi-Channel Memo Distribution Facility (MMDF). The script facility proved invaluable for debugging of *rpd*, and was essential when the leased line had not yet been installed.

The author expresses his gratitude to Jim Koda of the USC/Information Sciences Institute, who tirelessly supported the ISI end of the *split-gateway*, despite numerous system crashes and software changes. In addition, a note of thanks goes to Paul V. Mockapetris of USC/ISI who got the whole thing rolling by deciding that it was time for UCI to enter the networking world and join the ARPA Internet community.

[1] VMS is a trademark of Digital Equipment Corporation.

References

[CERF78] V.G. Cerf, "The Catenet Model for Internetworking", DARPA/IPTO, Internet Experiment Notebook 48, (July, 1978).

[GURW81] R.F. Gurwitz, "VAX-UNIX Networking Support Project — Implementation Description," Computer Systems Division, Bolt Beranek and Newman, Inc., Internet Experiment Notebook 168, (January, 1981).

[IP] "Internet Protocol," Request for Comments 791, Internet Protocol Transition Workbook, Network Information Center, SRI International, (September, 1981).

[JOY83] W.N. Joy, E. Cooper, R.S. Fabry, S.J. Leffler, K. McKusick, D. Mosher, "4.2BSD System Manual," Computer Systems Research Group, Technical Report Number 5, University of California, Berkeley, (July, 1983).

[LEFF83] S.J. Leffler, W.N. Joy, R.S. Fabry, "4.2BSD Networking Implementation Notes," Computer Systems Research Group, University of California, Berkeley, (July, 1983).

[OSI] "Reference Model of Open Systems Interconnection," ISO TC97/16 Document Number 227, (June, 1979).