IEN: 168

VAX-UNIX Networking Support Project
Implementation Description

Robert F. Gurwitz
Computer Systems Division
Bolt Beranek and Newman, Inc.
Cambridge, MA 02138

January, 1981

Table of Contents

1 Introduction

The purpose of this report is to describe the implementation of network software for the VAX-11/780 * running UNIX. ** This is being done as part of the VAX-UNIX Networking Support Project. The overall purpose of this effort is to provide the capability for the VAX to communicate with other computers via packet-switching networks, such as the ARPANET. Specifically, the project centers around an implementation of the DoD standard host-host protocol, the Transmission Control Protocol (TCP) [4]. TCP allows communication with ARPANET hosts, as well as hosts on networks outside the ARPANET, by its use of the DoD standard Internet Protocol (IP) [3]. The implementation is designed for the VAX, running VM/UNIX, the modified version of UNIX 32/V developed at the University of California, Berkeley [1]. This version of UNIX includes virtual paging capabilities.

In the following paragraphs, we will discuss some features and design goals of the implementation, and its organization.

2 Features of the Implementation

2.1 Protocol Dependent Features

2.1.1 Separation of Protocol Layers

The TCP software that we are developing for the VAX incorporates several important features. First, the implementation provides for separation of the various protocol layers so that they can be accessed independently by various applications. (1) Thus, there is a capability for access to the TCP level, which will provide complete, reliable, multiplexed, host-host communications connections. In addition, the IP level is also accessible for applications other than TCP, which require its internet addressing and data fragmentation/reassembly services. Finally, the implementation also allows independent access to the local network interface (in this case, to the ARPANET, whose host interface is defined in BBN Report No. 1822

---

* VAX is a trademark of Digital Equipment Corporation.
** UNIX is a trademark of Bell Laboratories.
(1) In this context, the terms application and user refer to any software that is a user of lower level networking services. Thus, programs such as FTP and TELNET can be considered applications when viewed from the TCP level, and TCP itself may be viewed as an application from the IP level.

[2]) in a "raw" fashion, for software which wishes to communicate with hosts on the local network and do its own higher level protocol processing.

2.1.2 Protocol Functions

Another feature of the implementation is to provide the full functionality of each level of protocol (TCP and IP), as described in their specifications [3,4]. Thus, on the TCP level, features such as the flow control mechanism (windows), precedence, and security levels will be supported. On the IP level, datagram fragmentation and reassembly will be supported, as well as IP option processing, gateway-host flow control (source-quenching) and routing updates. However, it is anticipated that some of these features (such as handling IP gateway-host routing updates, and IP option processing) will be implemented in later stages of development, after more basic features (such as TCP flow control and IP fragmentation/reassembly) are debugged.

2.2 Operation System Dependent Features

2.2.1 Kernel Resident Networking Software

There are several features of the implementation which are operating system dependent. The most important of these is the fact that the networking software is being implemented in the UNIX kernel as a permanently resident system process, rather than a swappable user level process.

This organization has several implications which bear on performance. The most obvious effect is that since the networking software is always resident, it can more efficiently respond to network and user initiated events, as it is always available to service such events and need not be swapped in. In addition, residence in the kernel removes the burden of the use of potentially inefficient interprocess communication mechanisms, such as pipes and ports, since simpler data structures, such as globally available queues, can be used to transmit data between the network and user processes. Kernel provided services, (e.g., timers and memory allocation) also become much easier and more efficient to use.

The large address space of the VAX makes this organization practical and allows the avoidance of expedients like the NCP split kernel/user process implementation, that have been necessary in previous UNIX networking software on machines with limited address space, like the PDP 11/70. It is hoped that the kernel resident approach will contribute to the speed and efficiency of this TCP.

2.2.2 User Interface

Use of the "traditional" UNIX file oriented user interface is another operating system dependent feature of this implementation. The user will access the network software by means of standard system file I/O calls: open, close, read, and write. This entails modification of certain of these calls to accommodate the extra information needed to open and maintain a connection. In addition, the communication of exceptional conditions to the user (such as the foreign host going down) must also be accommodated by extension of the standard system calls. In the case of open, for example, use of the call's mode field will be extended to accommodate a pointer to a parameter structure. In the case of exceptional conditions, the return code for reads and writes will be used to signal the presence of exceptional conditions, much like an error. An additional status call (ioctl) will be provided for the user to determine detailed information about the nature of the condition, and the general status of the connection.

In this way, the necessary additional information needed to maintain network communications will be supported, while still allowing the use of the functionality that the UNIX file interface provides, such as the pipe mechanism.

In the initial versions, this interface will be the standard UNIX blocking I/O mechanism. Thus, outstanding reads for data which has not been accepted from the foreign host, and writes which exceed the buffering resources of a connection will block. It is expected that the await/capacity mechanism, currently available for Version 6 systems, will be added to the VM/UNIX kernel in the near future. These non-blocking I/O modifications will be supported by the network software, relieving the blocking restriction.

3 Design Goals

Several design goals have been formulated for this implementation. Among these goals are efficiency and low operating system overhead, promoted by a kernel resident network process, which allows for reduced process and interprocess communication overhead.

Another goal of the implementation is to reduce the amount of extraneous data copying in handling network traffic. To achieve this, a buffer data structure has been adopted which has the following characteristics: intermediate size (128 bytes); low overhead (10 bytes of control information per buffer); and flexibility in data handling through the use of data offset and length fields, which reduce the amount of data copying required for operations like IP fragment reassembly and TCP sequence space manipulations.

The use of queueing between the various software levels has been limited in the implementation by processing incoming network data to the highest level possible as soon as possible. Thus, an unfragmented message coming from the network is passed to the IP and TCP levels, with queueing taking place at the device driver only until the message has been fully read from the network. Similarly, on the output side, data transmission is only attempted when the software is reasonably certain that the data will be accepted by the network.

Finally, it is planned that the inclusion of the network software will entail relatively little modification of the basic kernel code beyond that provided by Berkeley. The only modifications to kernel code outside the network software will be slight changes to the file I/O system calls to support the user interface described above. In addition, an extension to the virtual page map data structure in low core will be necessary to support the memory allocation scheme, which makes use of the kernel's page frame allocation mechanisms.

4 Organization

4.1 Control Flow

4.1.1 Local Network Interface

The network software can be viewed as a kernel resident system process, much like the scheduler and page daemon of Berkeley VM/UNIX. This process is initiated as part of network initialization. A diagram of its control and data flow is shown in Figure 1.

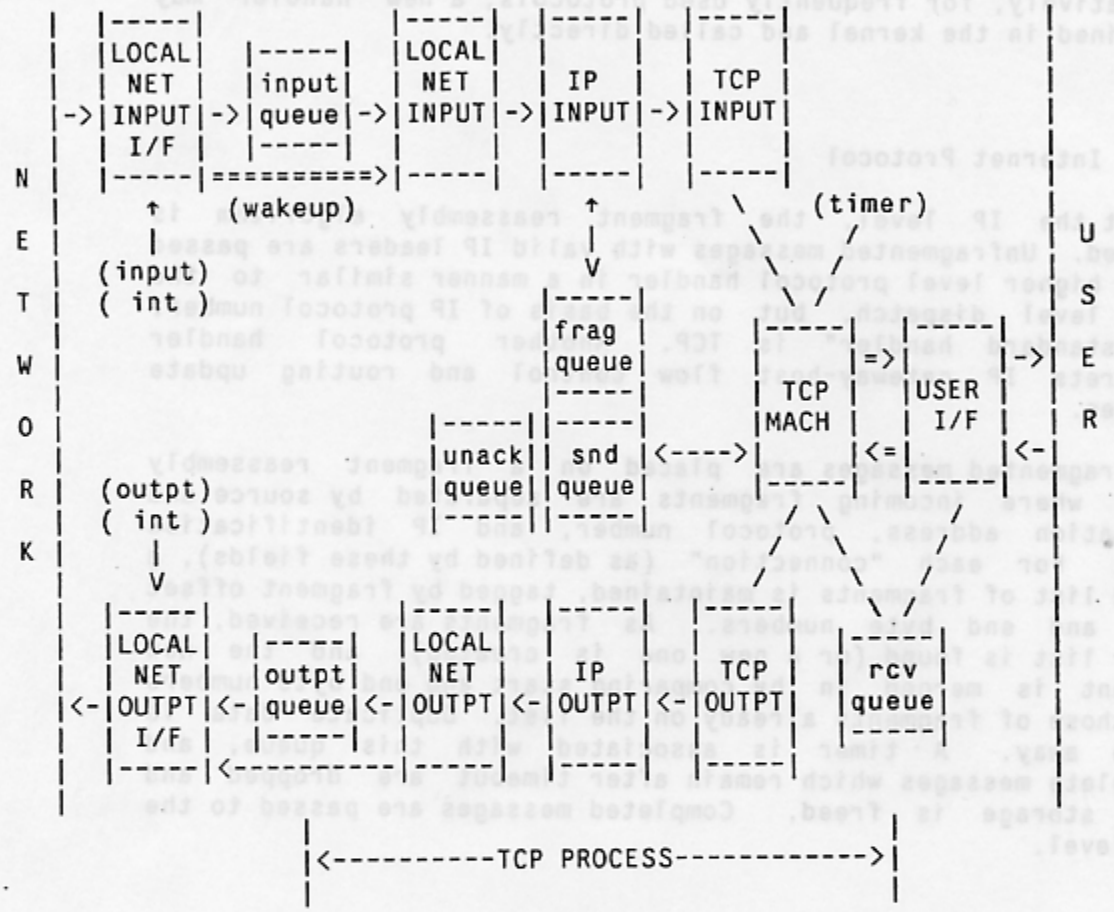

Figure 1 . Network Software Organization

Its main flow of control is an input loop which is activated (via wakeup) by the network interface device driver when an incoming message has been completely read from the network. (It can also be awakened by TCP user or timer events, described below.) The message is then taken from an input queue and dispatched on the basis of local network format (e.g., 1822 leader link number). ARPANET imp-host messages (RFNMs, incompletes, imp/host status)

are handled at this level. For other types of messages, the local network level input handler calls higher level "message handlers." The "standard message handler" is the IP input routine. Handlers for other protocols at this level (such as UNIX NCP) may be accommodated in either of two ways. First, a "raw message" service is available which simply queues data on specified links to/from the local network. By reading or writing on a connection opened for this service, a user process may handle its own higher level protocol communication. Alternatively, for frequently used protocols, a new handler may be defined in the kernel and called directly.

4.1.2 Internet Protocol

At the IP level, the fragment reassembly algorithm is executed. Unfragmented messages with valid IP leaders are passed to the higher level protocol handler in a manner similar to the lower level dispatch, but on the basis of IP protocol number. The "standard handler" is TCP. Another protocol handler interprets IP gateway-host flow control and routing update messages.

Fragmented messages are placed on a fragment reassembly queue, where incoming fragments are separated by source and destination address, protocol number, and IP identification field. For each "connection" (as defined by these fields), a linked list of fragments is maintained, tagged by fragment offset start and end byte numbers. As fragments are received, the proper list is found (or a new one is created), and the new fragment is merged in by comparing start and end byte numbers with those of fragments already on the list. Duplicate data is thrown away. A timer is associated with this queue, and incomplete messages which remain after timeout are dropped and their storage is freed. Completed messages are passed to the next level.

4.1.3 TCP Level

At the TCP level, incoming datagrams are processed via calls to a "TCP machine." This is the TCP itself, which is organized as a finite state machine whose states are roughly the various states of the protocol as defined in [4], and whose inputs include incoming data from the network, user open/close/read/write requests, and timer events. Input from the network is handled directly, passing through the above described

levels. User requests and timer events are handled through a work queue.

When a user process executes a network request via system call, the relevant data (on a read or write) is copied from user to kernel space (or vice versa), a work entry is enqueued, and the network process is awakened. Similarly, when timers associated with TCP (such as the retransmission timer) go off, timer work requests are enqueued and the network input process is awakened. Once awakened, it checks for the presence of completed messages from the network interface and processes them. After these inputs are processed, the TCP machine is called to handle any outstanding requests on the work queue. The network process then sleeps, waiting for more network input or work requests. Thus, the TCP machine may be called directly with network input, or awakened indirectly to check its work queue for user and timer requests.

After reset processing and sequence and acknowledgement number validation, acceptable received data is sequenced and placed on the receive queue. This sequencing process is similar to the IP fragment reassembly algorithm described above. Data placed on this queue is acknowledged to the foreign host. Received data whose sequence numbers lie outside the current receive window are not processed, but are placed on an unacknowledged message queue. The advertised receive window is determined on the basis of the remaining amount of buffering allocated to the connection (see below). When buffering becomes available, data on the unacknowledged message queue is then processed and placed on the receive data queue.

On the output side, TCP requests for data transmission result in calls to the IP level output routine. This routine does fragmentation, if necessary, and makes calls on the local network output routine. Outgoing messages are then placed on a buffering queue, for transmission to the network interface by the device driver. In data transmission, an attempt is made to ensure that data moving from the highest level (TCP), will not be sent unless there is reasonable certainty that the lower levels will have the necessary resources to accept the message for transmission to the network.

All data to be sent is maintained on a single send queue, where data is added on user writes, and removed when proper acknowledgement is received. Whenever the TCP machine sends data, a retransmission timer is set, and the sequence number of the first data byte on the queue is saved. After initial transmission the sequence number of the next data to send is advanced beyond what was first sent. If the retransmission timer

goes off before that data is acknowledged, the sequence number of the next data to send is backed up, and the contents of the send buffer (for the length determined by the current send window) is retransmitted, with the ACK and window fields set appropriately. The retransmission timer is set with increasingly higher values from 3 to 30 seconds, if the saved sequence number does not advance.

A persistence timer is also set when data is sent. This allows communication to be maintained if the foreign process advertises a zero length window. When the persistence timer goes off, one byte of data is forced out of the TCP.

4.2 Buffering Strategy

As mentioned earlier, all data is passed from the network to the various protocol software layers in intermediate sized (128 byte) buffers. The buffers have two chain pointers, a data offset, and a data length field (see Figure 2). As data is read from the network or copied from the user, multiple buffers are chained together. Protocol headers are also held in these buffers. As messages are passed between the various software levels, the offset is modified to point at the appropriate header. The length field gives the end of data in a particular buffer. This offset/length pair facilitates merging of messages in IP fragment reassembly and TCP sequencing.

The allocation of these buffers is handled by the network software. Buffers are obtained by "stealing" page frames from the kernel's free memory map (CMAP). In VM/UNIX, these page frames are 1024 bytes long, and thus have room for eight 128 byte buffers. The advantage of using kernel paging memory as a source of network buffers is that their allocation can be done totally dynamically, with little effect on the operation of the overall system. Buffers are allocated from a cache of free page frames, maintained on a circular free list by the network memory allocator. As the demand for buffers increases, new page frames are stolen from the paging freelist and added to the network buffer cache. Similarly, as the need for pages decrease, free pages are returned to the system. To minimize fragmentation in buffer allocation within the page frames, the free list is sorted. When no more pages are available for allocation, data on the IP reassembly and TCP unacknowledged data queues are dropped, and their buffers are recycled.

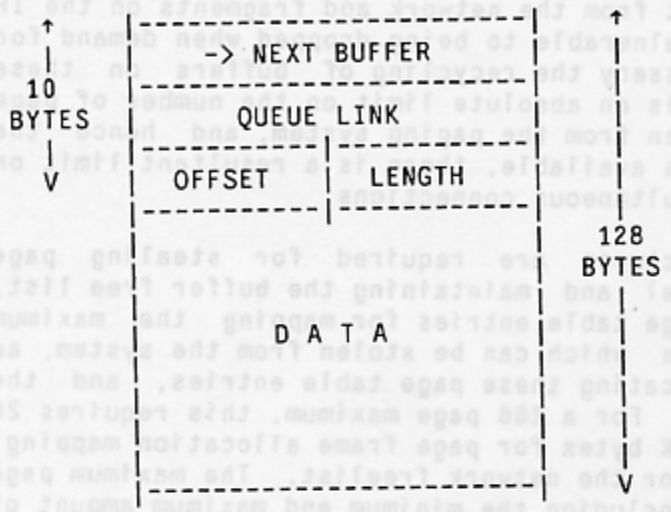

Figure 2 . Layout of a Network Buffer

The number of pages that can be stolen from the system is limited to a moderate number (in practice 64-256, depending on network utilization in a particular system). To enforce fairness of network resource utilization between connections, the number of buffers that can be dedicated to a particular connection at any time is limited. This limit can be varied to some small degree by the user when a connection is opened. Thus, a TELNET user may open a connection with the minimum 1K bytes of send and receive buffering; while an FTP user, anticipating larger transfers, might desire up to 4K of buffering. The effect of this connection buffering allocation is to place a limit on the amount of data that the TCP may accept from the user for sending before blocking, and the amount of input from the network that the TCP may acknowledge. Note that in receiving, the network software may allocate available buffers beyond the user's connection limit for incoming data. However, this data is considered volatile, and may be dropped when buffer demands go higher. Incoming data is acknowledged by TCP only until the user's connection buffer limit is exhausted. The advertised TCP flow control window for a connection is set on the basis of the remaining amount of this buffering.

Thus, the network software must insure that it has enough buffering for 1) its own internal use in processing data on the IP and local network levels; 2) retaining acknowledged TCP data

that have not been copied to user space; and 3) retaining data accepted by the TCP for transmission which have not yet been acknowledged by the foreign host TCP. Other data, such as unacknowledged TCP input from the network and fragments on the IP reassembly queue are vulnerable to being dropped when demand for more buffers makes necessary the recycling of buffers on these queues. Since there is an absolute limit on the number of page frames that may be stolen from the paging system, and hence the total number of buffers available, there is a resultant limit on the total number of simultaneous connections.

Several data structures are required for stealing page frames from the kernel and maintaining the buffer free list. These include enough page table entries for mapping the maximum number of page frames which can be stolen from the system, an allocation map for allocating these page table entries, and the free page list itself. For a 256 page maximum, this requires 2K bytes of page tables, 1K bytes for page frame allocation mapping, and another 1K bytes for the network freelist. The maximum page parameter and others, including the minimum and maximum amount of buffering that the user may specify are modifiable constants of the implementation.

4.3 Data Structures

Along with the data structures needed to support the buffer management system, there are several others used in the network software (see Figure 3). The focus of activity is the user connection block (UCB), and the TCP control block (TCB). The UCB is allocated from a table on a per connection basis. It holds non-protocol specific information to maintain a connection. This includes a pointer the UNIX process structure of the opener of a connection, (2) a pointer to the foreign host entry for the peer process's host, a pointer to the protocol-specific connection control block (for TCP, the TCB), pointers to the user's send and receive data buffer chain, and miscellaneous flags and status information. When a network connection is opened, an entry in the user's open file table is allocated, which holds a pointer to the UCB.

For TCP connections, a TCB is allocated. All TCBs are chained together to facilitate buffer recycling. The TCB contains a pointer to the corresponding UCB, a block of sequence number variables and state flags used by the TCP finite state

---

(2) For details on data structures specific to UNIX, see [5].

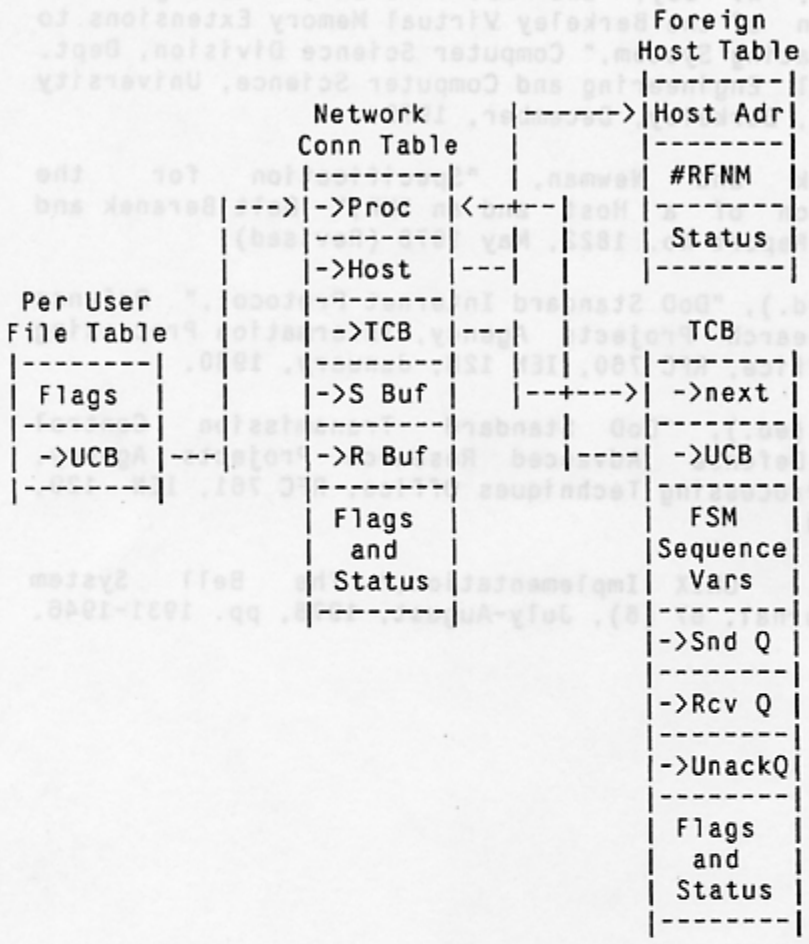

Figure 3 . Network Data Structures

machine, pointers to the various TCP data queues, and flags and state variables. Protocols other than TCP would have their own control blocks instead of the TCB. For the "raw" local network and IP handlers, all necessary information is kept in the UCB.

Finally, there is a foreign host table, where entries are allocated for each host that is part of a connection. The entry contains the foreign host's internet address, the number of outstanding RFNM's for 1822 level host-imp communication, and the status of the foreign host. Entries in this table are hashed on the foreign host address.

5 References

[1] Babaoglu, O., W. Joy, and J. Porcar, "Design and Implementation of the Berkeley Virtual Memory Extensions to the UNIX Operating System," Computer Science Division, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, December, 1979.

[2] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," Bolt Beranek and Newman Inc., Report No. 1822, May 1978 (Revised).

[3] Postel, J. (ed.), "DoD Standard Internet Protocol," Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC 760, IEN 128, January, 1980.

[4] Postel, J. (ed.), "DoD Standard Transmission Control Protocol," Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC 761, IEN 129, January, 1980.

[5] Thompson, K., "UNIX Implementation," The Bell System Technical Journal, 57 (6), July-August, 1978, pp. 1931-1946.