A Performance Analysis of TCP/IP and UDP/IP Networking Software for the DECstation 5000

Jonathan Kay (jkay@cs.ucsd.edu)

Joseph Pasquale[1] *(pasquale@cs.ucsd.edu)*

Computer Systems Laboratory

Department of Computer Science and Engineering

University of California, San Diego

La Jolla, CA 92093-0114

Abstract

Modern workstations increasingly rely on distributed software such as NFS and NIS, yet the speed of networking software is not improving as rapidly as the workstation and networking hardware, leading to a network software bottleneck. We present detailed measurements of various components of the TCP/IP and UDP/IP protocol stacks on a DECstation 5000/200 running Ultrix 4.2a. Measurements are by layer (i.e. socket, transport, IP, data-link) and by function (i.e. checksum computation, data copying, buffer management, protocol processing, and operating system interaction), with further breakdowns within each category. We show that checksum computation and data transfers dominate component times for a real LAN workload, that using large packet MTUs (maximum transmission units) is very important to achieving high throughput, and that given the distribution of component times for small sized messages, it will be difficult to improve latency. TCP and UDP time breakdowns are shown to be quite similar, suggesting that "lightweight" transport protocols are not likely to greatly decrease processing time. Finally, analytical models for network software processing times are presented.

1.0 Introduction

Network software speed is not increasing as rapidly as that of workstation CPUs. The goal of this study is to determine how various components of network software contribute to this bottleneck. We instrument the TCP/IP and UDP/IP protocol stacks since almost all of the traffic on the networks we measured is comprised of messages based on these protocols. We analyze network software component processing time by layer and by operation. By operation, we present detailed measurements for processing times for major categories of operations, including checksum computation, data movement, protocol processing, buffer management, operating system functions, data structure manipulations, and error checking. Other studies have

1. This work was supported in part by grants from DEC, IBM, NCR, NSF, TRW, and UC MICRO.

shown some of these operations to be expensive [3, 6-8, 11, 22]. By layer, we measure the individual processing times of the socket, TCP and UDP, IP, link, and device driver software.

We perform this study for the following reasons. First, the performance characteristics of current RISC-based workstations (e.g. the DECstation 5000) and fast highly reliable optic-fiber LANs (e.g. FDDI) are different from those used in past studies [3, 8]. Second, we wanted to experiment with a variety of parameters, allowing us to understand both latency and throughput performance characteristics using a wide range of packet sizes. We include latency, as most a large portion of LAN traffic using TCP and UDP protocols is comprised of messages smaller than 200 bytes [5, 10, 16], and increasing numbers of RPC-based applications [19-21] are likely to be more constrained by latency than by bandwidth. Many high throughput applications are increasingly constrained by high latencies and low packet sizes. High latencies, which often derive a large contribution from router network software, require larger TCP windows [12]. Furthermore, in the wide area Internet arena, the current maximal packet size across wide area networks is largely frozen by [18] at 527 bytes, a relatively small value when one compares this value to maximum Ethernet and FDDI packet sizes.

However, processing time for large packets is certainly important; large packets consume more resources than small packets, and large packets are frequently produced under situations of greater processing load, e.g. NFS service. Nonetheless, confining one's interest to optimal high throughput situations seems somewhat analogous to restricting one's interest in workstation performance to peak MFLOPS. To gain some sense of network software performance under a wide range of conditions, including typical LAN conditions, we measured network software processing times for a wide range of message sizes, and took a packet trace and used the results to drive network software under measurement.

2.0 Experiments

We connect an HP 1652B Logic Analyzer to the DECstation TURBOchannel to obtain software processing time measurements with a resolution of 40 nanoseconds (which is the DECstation clock cycle time). Special C preprocessor macros are placed at the beginning and end of the source code of each operation of interest. Each time a macro executes, a pattern plus an event identifier is sent over the DECstation's I/O bus. The logic analyzer is programmed to recognize the pattern and store the event identifier, along with a timestamp. The measurement software causes minimal interference, generating overhead of less than 1% of the total network software processing time.

The experimental system consists of two workstations connected by an FDDI network with no other workstations and no network traffic other than that generated by the experiment. An experiment consists of one workstation sending a message to the *system under test*, which then sends the same message back. All measurements are made on the system under test, which is executing a probed kernel and is hooked up to the logic analyzer. The experiments are repeated 100 times at the same message size to obtain statistical significance in the results. That experiment, in turn, is repeated for 40 message sizes evenly spaced from 1 byte to 8192 bytes. Note that the described workload design has the result that TCP acknowledgments (ACKs) are very nearly always piggybacked.

We also drive a series of tests using packet sizes constructed by observing a real workload. Using the program `tcpdump`, we obtained a trace of packets from an FDDI network which is used primarily as a LAN for workstations. This packet trace is used to drive our test software to produce packets of the sizes actually seen in the trace, in the order that they appear in the trace. In support of our workload decisions, no messages larger than 8192 bytes were seen in the trace, and most ACKs (about 2/3) were piggybacked.

3.0 Results

Several analyses of our results are presented. Initially we present timing information for the various operations and layers in the network code. Then the operation times are analyzed several different ways: operation times distinguished by sending and receiving; operation times specifically for small messages; operation times resulting from the real packet trace driver. The understanding of operation times is used to develop a simple but accurate model for processing costs in the network software. Finally, an analysis of various operation times by throughput is presented.

We note two conventions used in this paper. One is that we refer to throughputs in terms of megabytes/second (1,000,000 bytes/sec) rather than the megabits/second; the maximum Ethernet throughput is 1.25 megabytes/sec and the maximum FDDI throughput is 12.5 megabytes/sec. The other convention is that our usage of the term “link layer” refers to layers of code below the Internet Protocol, yet above the device driver, such as ARP (i.e. Address Resolution Protocol) and Ethernet encapsulation. We are aware that this is somewhat different from the spirit of the link layer as described by the OSI [4], yet it is not altogether unusual within the Internet community [1].

3.1 Analysis by Operation

We present the total message processing time for various components of network software when receiving and then sending the same-sized message by operation. Because we measured a large number of different operations, we grouped the operations in the following major categories: checksumming, data movement, network buffer management, protocol processing, operating system overhead, data structure manipulation, and error checking. An additional category, Other, is the amount of processing time we have not been able to account for. Checksum is a single operation, the Internet checksum routine [2]. Data movement includes all operations that cause data to be moved from one place to another. Buffer management includes allocation and freeing of “mbufs”, the major data structure used to store messages for Berkeley UNIX-based network software [13]. Protocol processing is protocol-specific work such as setting header fields and maintaining state. Operating system overhead includes synchronization overhead and other general OS support functions. Data structure manipulation comprises manipulations of various data structures other than mbufs. Error checking the category of checks for user and system errors, and includes parameter checking on socket system calls.

Figures 1a-d show the per-message send+receive processing times for the various categories of operations for TCP and UDP messages, respectively. For example, the checksum time of 359 microseconds for a 1024 byte message as shown in Figure 1 refers to the total time spent in the six calls to the checksum routine: two header and one data checksum computations while receiving a 1024 byte message, and the same for sending the same message back. Since the maximum FDDI frame size is 4352 bytes, IP must break larger messages into two FDDI frames when sending and reassemble them when receiving (fragmentation/defragmentation). Figures 2a and 2b show the breakdown as accumulated percentages of total processing time.

We were surprised to find that the overall breakdowns for sending and receiving TCP/IP and UDP/IP messages are very similar. Perhaps the largest difference is that the Protls category is slightly larger for TCP than for UDP, most noticeable in Figures 2a and 2b. This is because, although TCP is in many ways the most complicated and difficult to understand portion of the TCP/IP implementation, it is only a relatively small part of the executed layers of network software. Sending or receiving a TCP/IP message involves calling IP, the socket layer, the FDDI driver, and numerous support routines.

.

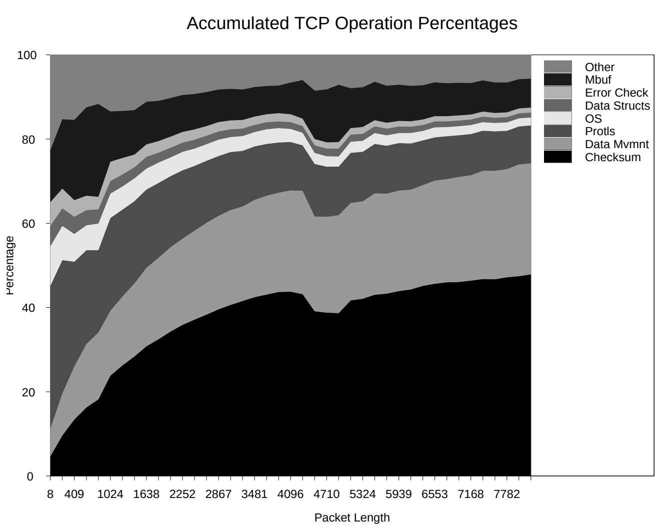

Figure 2a

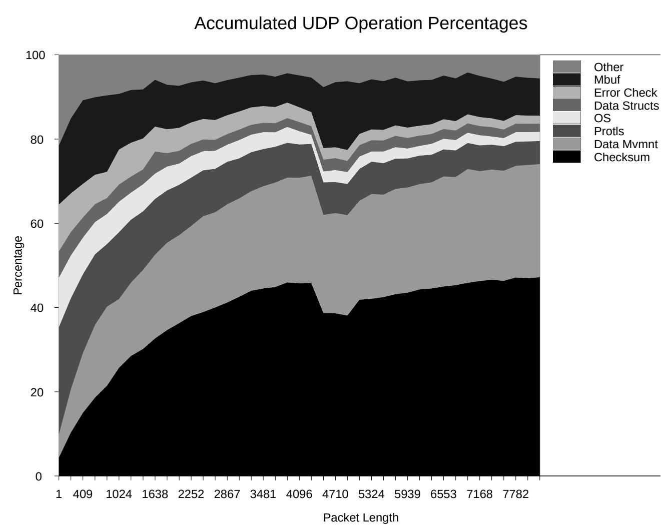

Figure 2b

Checksum and Data Mvmnt clearly dominate processing time for large messages. The situation is not so clear for small messages (to be discussed later in the section of small message analysis). These graphs suggest a division into two classes of operations: operations that touch data and whose expense scales linearly with message length, and operations that are closer to being constant in cost.

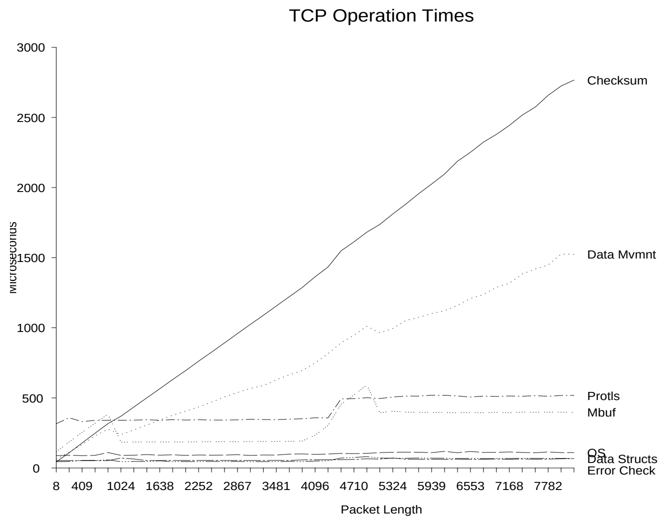

Figure 1a

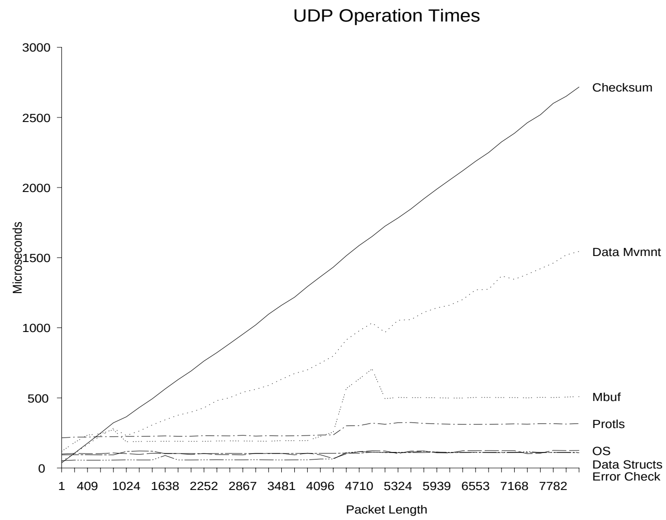

Figure 1b

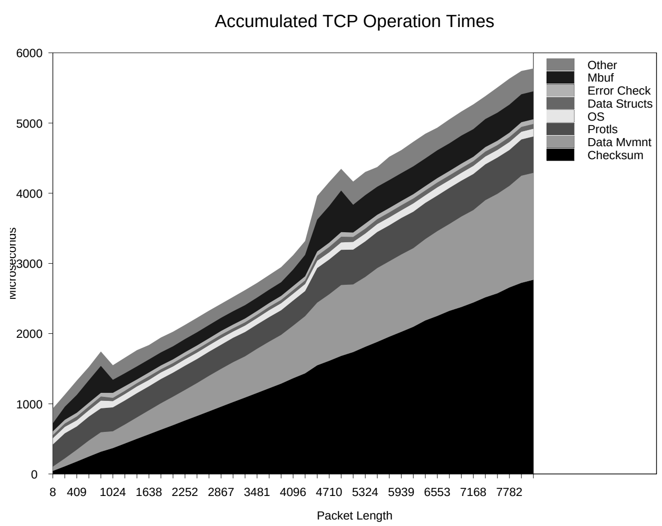

Figure 1c

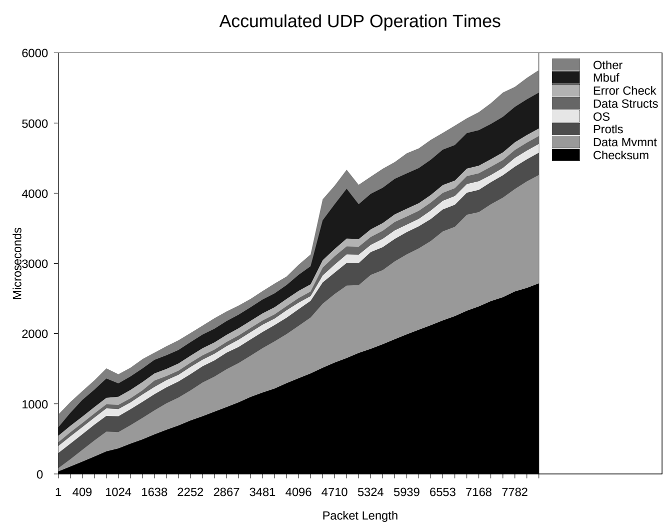

Figure 1d

3.1.1 Touching Data

Checksum and Data Mvmnt are the operations that touch each byte of the entire packet; thus, their operation times increase with message size. The data-dependent routines are the same for both TCP and UDP.

As can be seen in Figures 1 and 2, for large packets, Checksum is the dominant operation, taking about 40% of the total processing time. If one is to improve throughput using large packets, the implementation of the Checksum routine is a good place to start. In fact, we have reimplemented the routine, optimizing its performance by unrolling the inner loop and minimizing memory accesses (by moving words rather than bytes, an important optimization for RISC processors). This resulted in a doubling of the speed of the Checksum routine, and consequently an overall performance improvement of 20% in large packet throughput [23]. Note that the software tested in the experiments described in this paper do not include these performance optimizations.

Data Mvmnt is the next largest category of operations for large packets; see Figures 3a and 3b for a breakdown of this category. It includes three operations: Device Copy, Usr-Krnl Cpy, and Cache Clear. Device Copy is the amount of time needed to copy each message to buffers in the FDDI controller. Usr-Krnl Cpy is the amount of time spent copying incoming messages from kernel to user buffer and outgoing messages from user to kernel buffers. Cache Clear is the amount of time used to insure cache consistency when the FDDI controller DMAs incoming packets into memory. Time to copy data between the device and memory dominates the other times, with user-kernel copying time a close second. In fact, Device Copy and Krnl Cpy roughly do the same amount of work, but caching improves the latter’s time.

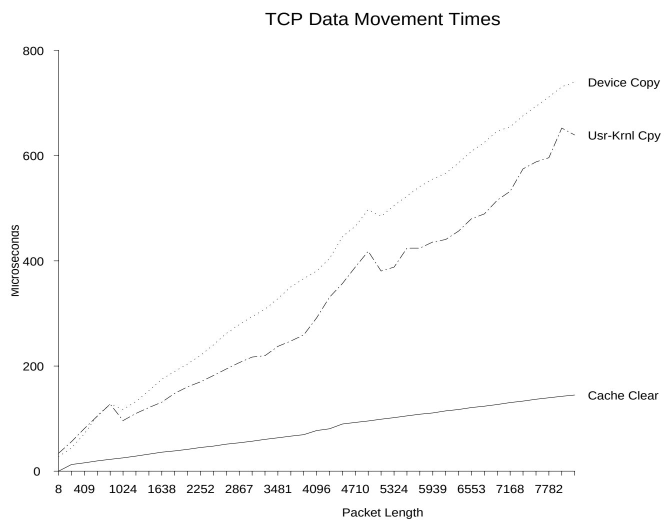

Figure 3a

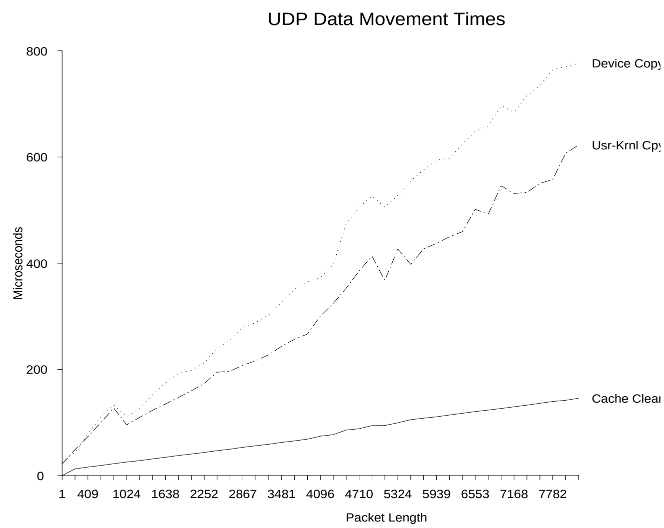

Figure 3b

3.1.2 Mbufs

The remaining categories do not rise directly with message size. Mbuf is one of the largest of those. Berkeley Unix based network subsystems buffer network data in a data structure called an ‘mbuf.’ Mbufs support a number of operations, the most costly of which are included in this analysis. Figures 4a and 4b contain breakdowns of mbufs into mbuf allocation and deallocation (Mbuf Alloc) and all other mbuf operations (Mbuf Misc).

It is helpful to understand certain aspects of mbuf design to understand the many mbuf-related graphs in this paper. An mbuf is a 128-byte structure. It can either directly store up to 108 bytes of data or a pointer to a 4096-byte page, which is used to store data of size greater than 108 bytes. If there is a pointer to a page, the combination is called a ‘cluster’ mbuf and no data is stored in the header. Mbufs contain linked list pointers so they can be easily strung together into an ‘mbuf chain’ to contain longer packets, or to prepend protocol headers to mbuf chains.

Mbufs use a complex allocation algorithm. To reduce internal fragmentation, the socket sending code allocates a string of up to 10 small mbufs if the message is less than 1024 bytes long, calling a memory allocation routine for each mbuf; this explains the humps below 1024 bytes in Figures 4a and 4b. For messages larger than 1024 bytes but less than a page, a

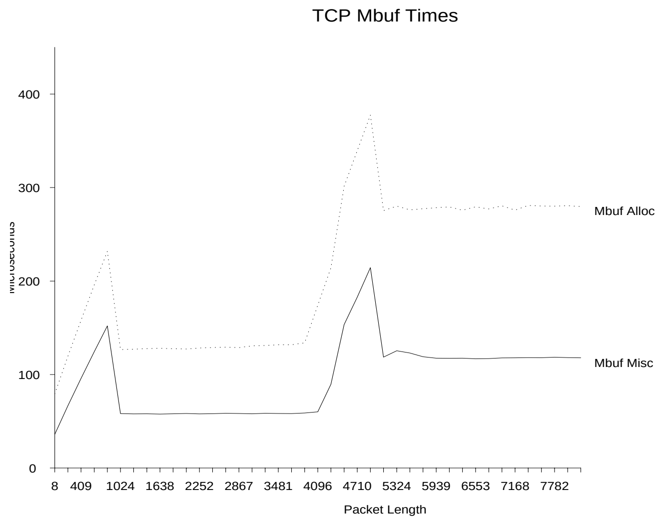

Figure 4a

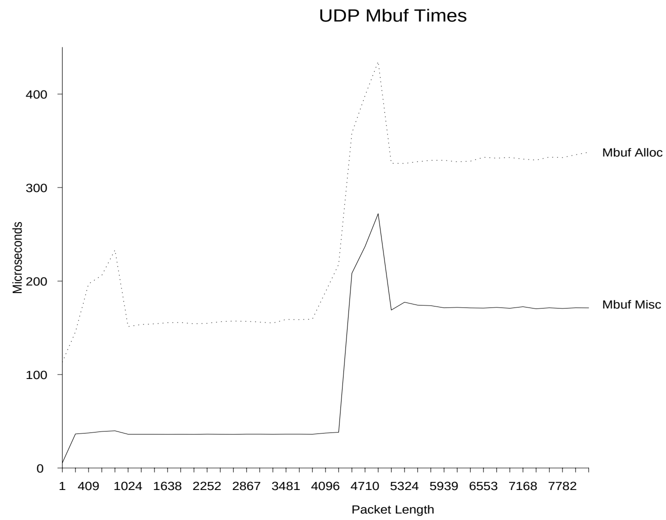

Figure 4b

cluster mbuf is allocated; this causes two calls to the memory allocator, one for the mbuf and another for the page. In similar fashion, another string of mbufs is allocated for messages larger than a page but smaller than 5120 bytes, whereas a second cluster mbuf is allocated for messages larger than 5120 bytes. The packet reception code in the FDDI driver, on the other hand, allocates a small mbuf for packets smaller than 108 bytes, a cluster mbuf for packets larger than 108 bytes but less than 4096 bytes, a cluster mbuf and a small mbuf for packets larger than 4096 bytes but less than 4204 bytes (4096 + 108), and two cluster mbufs for packets larger than 4204 bytes. The maximum transmission unit (MTU) for FDDI is 4352 bytes, so TCP fragments messages larger than 4312 bytes (FDDI MTU - sum of TCP and IP header lengths) into multiple packets. Thus, two cluster mbufs are always allocated upon packet reception as a result of messages larger than 4312 bytes, plus whatever allocations result from applying the same algorithm on the second packet necessary to contain messages up to the 8192 byte size limit of our experiments. This explains the two plateaus in Figures 4a and 4b. Notice that in the worst case, at 5120 bytes, the memory allocator, and consequently the deallocator as well, are each called 16 times, for a total of 275 microseconds.

The Mbuf allocation strategy has a significant effect on network software performance which is not reflected solely in the amount of time spent allocating memory. Many parts of the network code do more work when there are many small mbufs rather than a few large ones; probably the most common example is any code that loops through chains of mbufs; the number of loop iterations depends upon the number of mbufs in the chain. That is why humps in our timing graphs just below 1024 and 5120 bytes pervade many measurements other than the mbuf operations.

3.1.3 Protocol Processing

A breakdown of the Protls category is shown in Figures 5a and 5b. Interface Protl is in the device driver layer, Link Protl is part of the IEEE 802 encapsulation layer, IP Protl is in the IP, and TP Protl is within TCP. Arp is the entire Address Resolution Protocol [17], Demux is the operation of finding a protocol control block, given a TCP header (in_pcblookup), and

PCB (Dis)connect is the operations of checking that a route already exists for a connection, setting up the protocol control block to reflect current connection state properly (in_pcbconnect and in_pcbdisconnect).

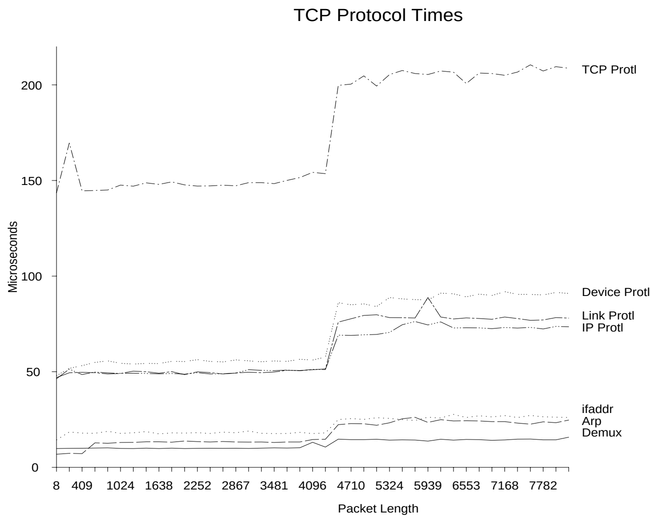

Figure 5a

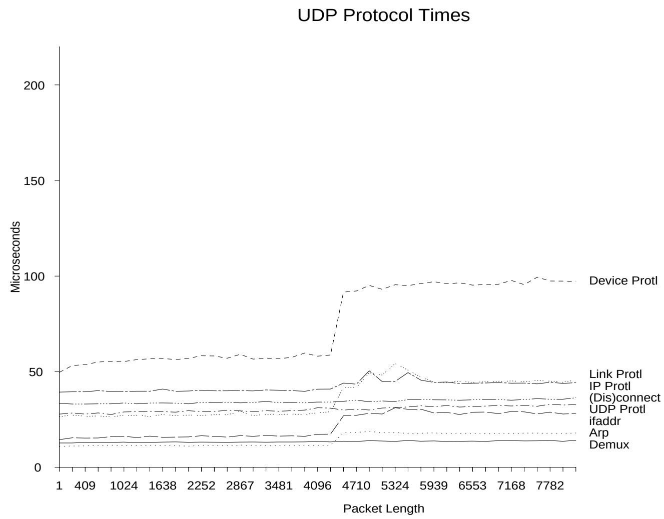

Figure 5b

The most expensive operation, Interface Protl, is device driver code, mostly miscellaneous communication with the interface board. We believe that most of the time spent in Link Protl is from copying small data structures using bcopy. Notice that in Figures 5a and 5b, Interface Protl, Link Protl, and IP Protl jump at the FDDI MTU. This is because when messages are fragmented into multiple packets, these layers are called twice as often, since fragmentation occurs in the TCP layer.

At first glance, it may be surprising that Demux is so small, under 20 microseconds, given reports that this operation is a bottleneck. This is because of our testing environment, which has only a single connection, and hence a single entry on the list of protocol control blocks, exists for the duration of the tests.

3.1.4 Operating System Overhead

The OS category consists of operating system support functions. Figures 6a and 6b show the breakdown of operations for this category. Sleep is the kernel call a process uses to block itself. Socket Misc is the miscellaneous processing of the socket layer. Wakeup is the call used to awaken that process. Proc Restart is the time needed for the sleeping process to start running again. Sched Soft Intr is the operation of scheduling a soft interrupt to process the incoming packet, and Softint Handler is the entry point called by the software interrupt which dequeues incoming packets and calls IP to handle them. Perhaps the most interesting aspect of this category is that the various transfer-of-control operations are so cheap.

3.1.5 Data Structure Manipulations

The Data Structs category consists of manipulations of various data structures that are not so expensive as to be worth close individual scrutiny as was the case with mbufs. Figures 7a and 7b show the breakdown of operations for this category.

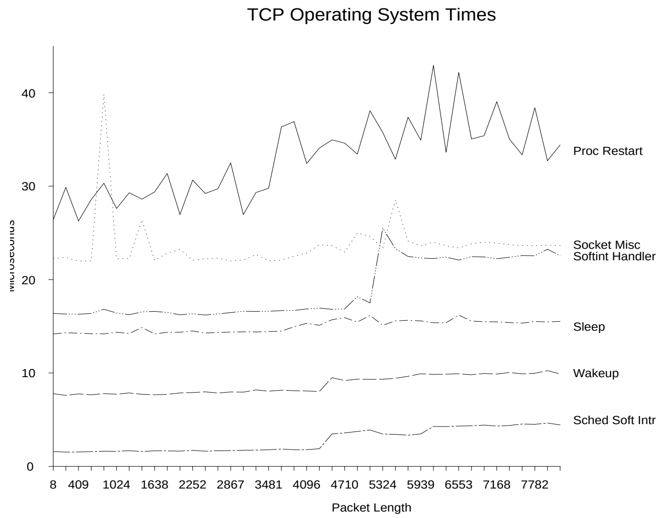

Figure 6a

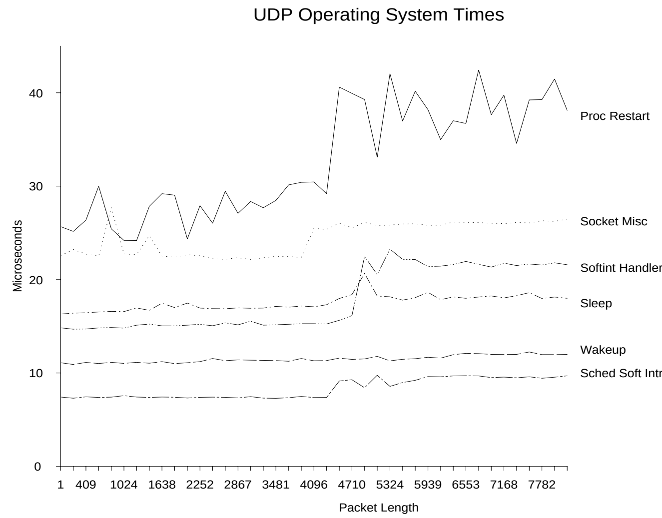

Figure 6b

The socket buffer is the data structure in which a limited amount of data is enqueued either for or by the transport protocol. The Device Queue is the data structure in which outgoing data is enqueued by the link layer until the network controller is prepared to process it, and ingoing data is enqueued by the device driver until IP is ready to process it in a software interrupt. The Defrag Queue is the data structure used to defragment IP packets. The shapes of the Defrag Queue curves for TCP and UDP require explanation. Since TCP goes to some effort to avoid IP fragmentation, it is surprising that the code checking the defragmentation queue is called at all; this reflects a place in the IP code where a check for matching fragments could have been avoided. Yet Defrag Queue in UDP is essentially zero when there are no fragments. It is not surprising that UDP Frag Queue goes high once UDP fragments occur.

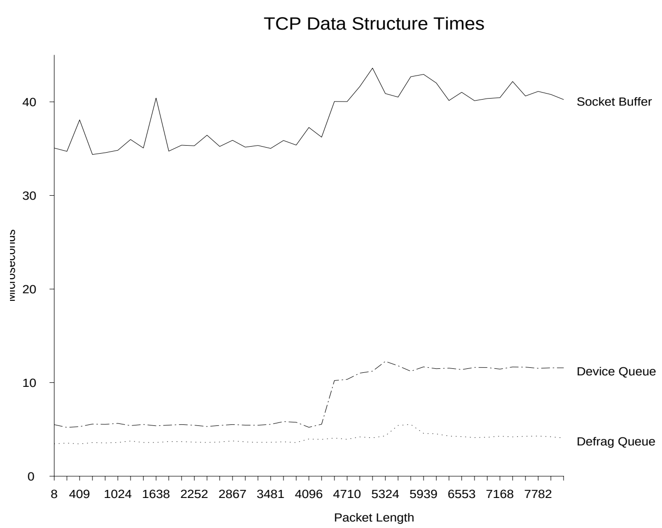

Figure 7a

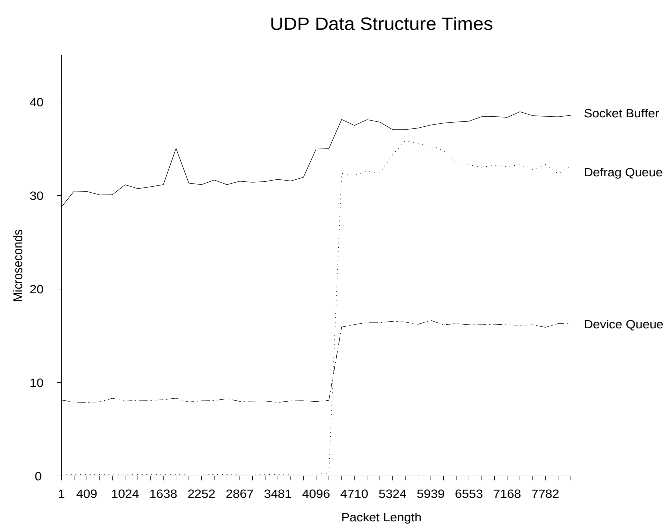

Figure 7b

3.1.6 Error Checking

Error Check is the category of checks for user and system errors. Figures 8a and 8b show the breakdown of operations for this category. Skt Errchk is assorted checks for errors within the socket layer. Syscall Argchk is checking specifically for incorrect user arguments to the system calls used to send and receive messages. Interestingly, Syscall Argchk is actually rather large, especially for the UDP workload. Most of this is because of time spent verifying that a user's buffer is of the correct size. Its largeness under UDP is an artifact of the workload implementation, though we note that user-level UDP receive buffers need to be the size of the largest expected message, and user-level TCP buffers may be of any size without loss of data. Interestingly, originally we used extremely large receive buffers so that we knew the receive sides of our workload generator would work no matter how the sending side was structured and no matter what protocol was in use; we cut down the size of the receive buffer once we saw that the single argument checking operation was dominating message reception time.

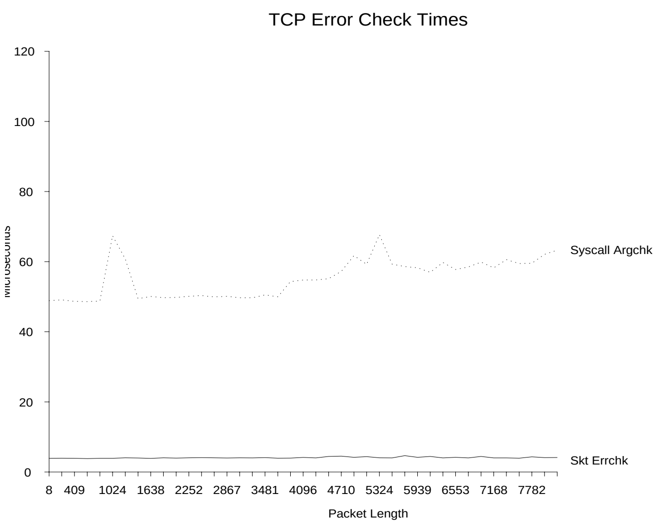

Figure 8a

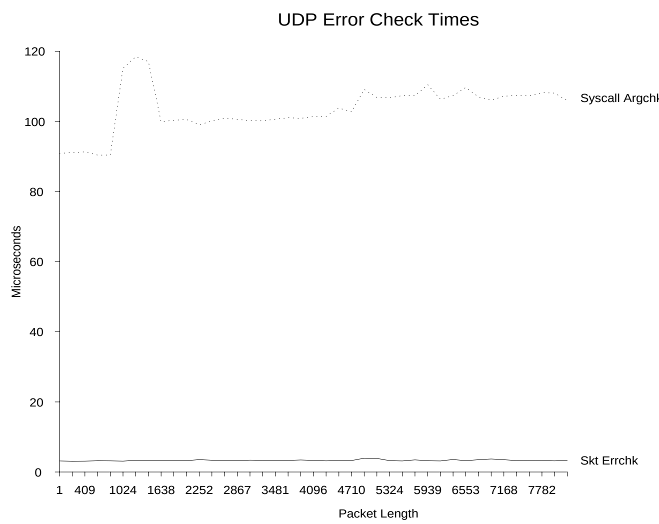

Figure 8b

3.1.7 Other

The Other category is the category of processing time we were unable to account for in this study. As can be seen from Figures 1 and 2, it is 22% in the worst case of 1-byte messages, quickly going down to under 20% and then gradually reducing to about 6%. Many operations are too small to measure without spending more time in the operation than the operation actually consumes. A very frustrating example of this was the symmetric multiprocessing locking mechanism called frequently in Ultrix; it was called literally hundreds of times per message, but each call consumed about the same amount of time as the error on the amount of time consumed by the two probe points necessary to measure, so it is not included in this analysis. Although, by definition, it could be anywhere in the software, it is our educated guess that most of the time unaccounted for is spent in the socket layer; that layer proved to be the most difficult to instrument accurately.

3.2 Analysis by Layer

Figures 9a and 9b show the per-message combined send and receive processing times for each layer of TCP/IP and UDP/IP, respectively, and Figures 23 and 24 show the accumulated processing times for each layer for TCP and UDP messages, respectively. These layer times are relatively easy to understand in light of the various operation times.

Clearly, the most expensive layer for large messages is the transport protocol layer, for both TCP and UDP; this is because each performs a data checksum; TCP is just slightly more expensive because the TCP Protl operation is larger than the UDP Protl operation, and because TCP does fragmentation in the TCP stack, and IP does fragmentation in the UDP/IP stack. That, in turn, brings up an interesting point: the increase in TCP when fragmentation begins in the TCP/IP stack is smaller than the rise in IP when fragmentation begins in the UDP/IP stack. That suggests the fragmentation algorithms used in the TCP implementation are more efficient than the algorithms used in IP.

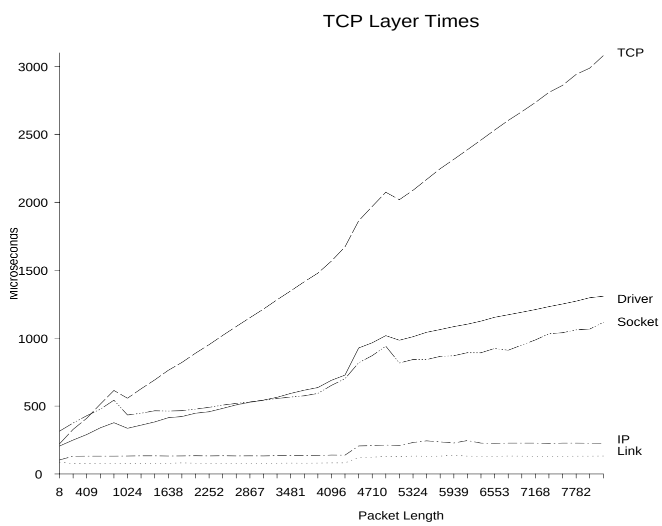

Figure 9a

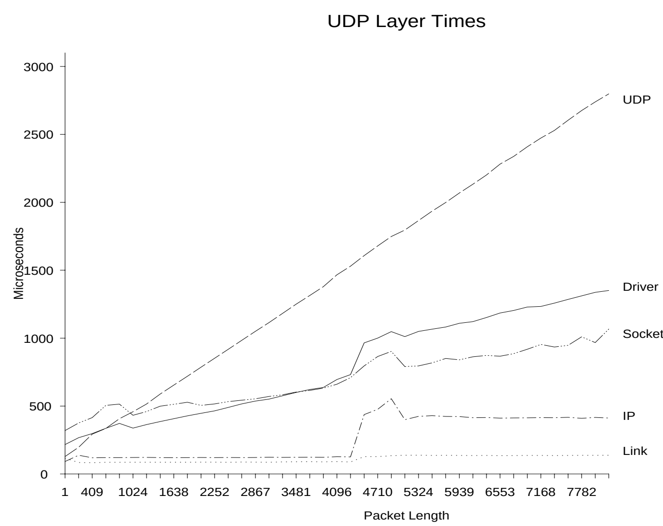

Figure 9b

Driver and Socket are close contenders to be the next largest time consumers. Driver is primarily large because of the device copy operation within it. The bottleneck within the socket layer is due to user/kernel copying, though the socket layer contains a fair amount of other processing, including the Error Check category and a sizable portion of Mbuf Alloc. IP and the link layer times are both relatively quite small, even when IP performs fragmentation.

3.3 Small Messages

Much attention has been focussed on high throughput networking with maximal packet sizes. However, most packets are actually rather small. Almost all packets sent over wide area networks are less than 576 bytes, the limit imposed by [18]. Even in our LAN packet trace, where no such limitation applies, most messages are indeed quite small, less than 200 bytes. We devote a section here to small packet performance, since performance characteristics and bottlenecks turn out to be different for small packets than for large ones. Message size 614 was chosen as the largest because it is the first measurement larger than

576 bytes, and because the data movement curves do not dominate until around 614 bytes (see Figure 1 and 2). Unlike large messages, neither checksum nor data movement dominates at all for TCP, nor for UDP until around 614 bytes, far larger than most messages. The protocol processing category actually dominates, consuming as much as 33.8% of single-byte TCP processing time. Figures 10a and 10b contain breakdowns of the protocols category for TCP and UDP. One should not conclude, though, that lightweight transport protocols are the key to low latency; in the largest case, TCP Protl only consumes 13.7% of the processing time. Reduction of TCP processing time would be useful, but it is questionable whether the performance improvement would be worth the major effort required. In fact, our observation is that operation times for small messages are well balanced. None ever rise above 16.3%, which is the value for checksum at the 614-byte message size. Large improvements in processing time of small messages are likely to be difficult to achieve.

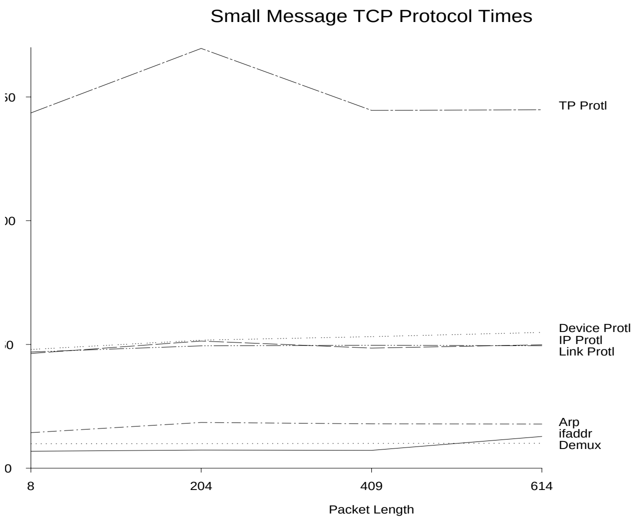

Figure 10a

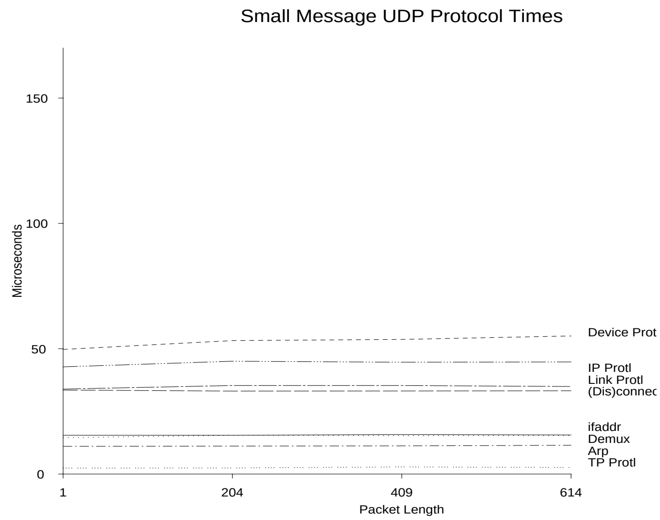

Figure 10b

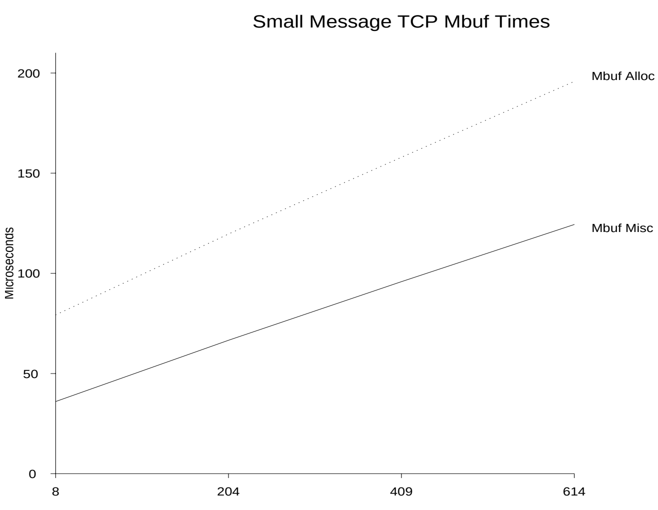

Figure 11a

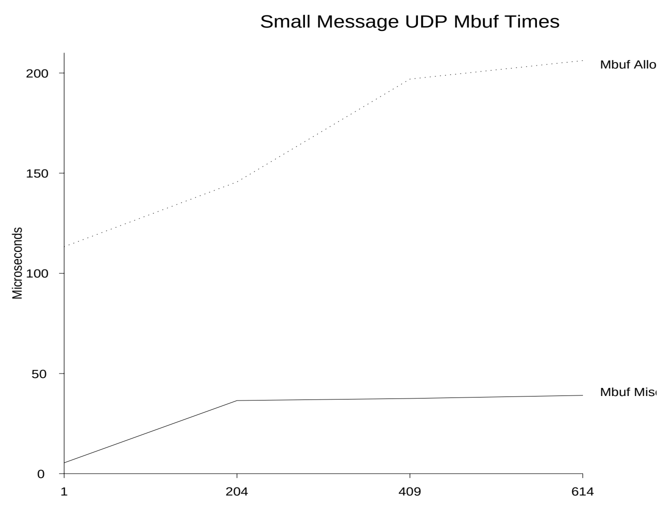

Figure 11b

The mbufs category is the next largest category after Protls, rising near and even above the protocol processing curve. Breakdowns for TCP and UDP are shown in Figures 11a and 11b. Mbuf allocation rises rapidly as more mbufs are allocated by the message sending code.

Processing times for single-byte messages are particularly interesting because they are the limiting factor for minimal latency performance, which is interesting in light of the increasing numbers of applications making use of RPC. Yet they are the best balanced, and thus difficult to improve significantly. The largest operation is TCP Protl, which consumes 13.7% of the processing time, and itself would require careful study to optimize. Since optimization does not look promising, and since latency is not decreasing in proportion to increases in processor speed, latency threatens to become an unsolvable bottleneck, at least at the level of functionality provided by the TCP/IP suite under Berkeley Unix. Distributed application designers would be well advised to batch requests whenever possible, or even avoid RPC-style interactions altogether.

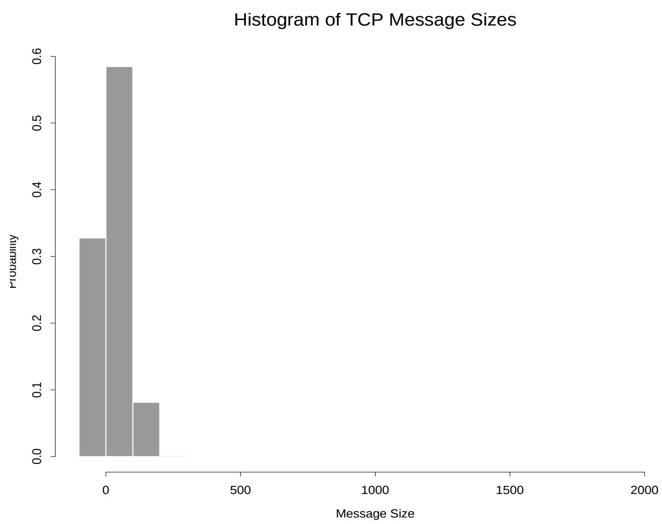

Figure 12a

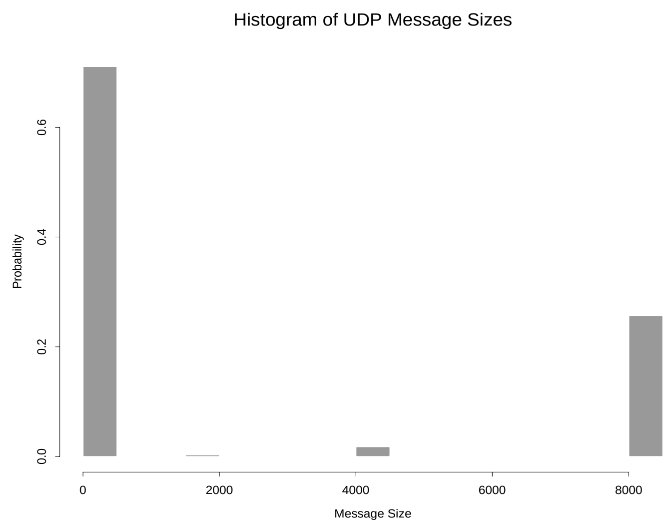

Figure 12b

3.4 Average Behavior

Knowing how much time is spent in which category for each packet is useful, but it does not tell one how much time is spent in what category under a real workload. One must also take into account the traffic patterns for that host, so we collected a packet trace from a computer science department's FDDI network of workstations. Experiments as described earlier were carried out, using message sizes as described by the trace. Figure 12a is a histogram of TCP packet sizes and Figure 12b is a histogram of UDP packet sizes. The reason for so many 8 kilobyte UDP packets is because of NFS. The complete lack of any large TCP packets at all was surprising at first. As a check, we observed a local Ethernet to see if this behavior was simply an artifact of the particular FDDI network we measured, but we found the same results. After some consideration, we concluded that the most common application that produces large messages is remote file distribution, which on workstations is done increasingly via NFS over UDP on most LANs. Thus, most of the large messages we observed in our environments are

UDP messages. Of course, there are applications that produce large TCP messages: FTP, X Window System image transfer, Network News Transfer Protocol, but these messages were relatively infrequent; out of sixteen hours worth of traces of various LANs, of those causes, only a single X Window System image transfer appears. Since a number of constraints limited us to running a few thousand messages through the probed network software, it is not unusual that no large TCP messages appear.

The mean TCP and UDP message sizes, 40 and 2340 bytes respectively, are quite different. However, the medians, which are important to consider for bimodal distributions, are 32 and 128 bytes, respectively. The low UDP median reflects the fact, visible in the histograms, that even most UDP messages are small (NFS status or directory reading messages).

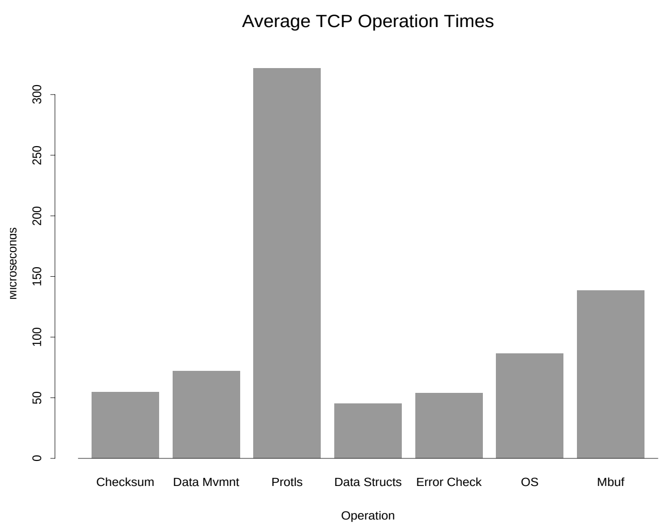

Figure 13a

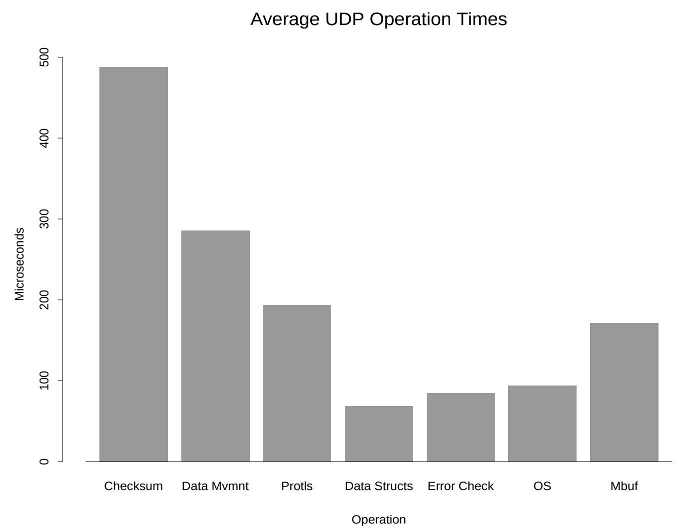

Figure 13b

3.4.1 Overall View of Operation Times

A breakdown of the processing times into categories appears in Figures 13a and 13b. Using medians for message sizes, the total processing times for TCP and UDP are very similar, at 982 and 1110 microseconds, respectively. The TCP total processing time is close to the processing time for single-byte messages and the UDP total processing time is close to the processing time for 409-byte messages. Despite the similarity in total processing time, the profile of contributors to the two times differ greatly. The TCP profile is very much that of small messages, with protocol and mbuf processing, both “constant” costs, dominating. The UDP breakdown, on the other hand, looks more like the breakdown for a large message; the message-size-dependent operations, checksum and data movement, dominate. Closer analysis of the two protocol profiles is done separately, since they are so different.

Since the TCP breakdown is fundamentally a breakdown for small messages, it obeys the observation made in the section on small messages that processing time components are balanced. Only two operations consume as much as 10% of the processing tim, TCP Protls and Mbufs Alloc. TCP Protl consumes 128 microseconds, which is 13% of the processing time.

The next largest operation time is Mbuf Alloc, at 92.8 microseconds or 9.5%. All other individual operations are performed in less than 50 usec, or 5%.

The UDP timing breakdown is very much that of large messages. Checksum dominates at 488 microseconds or 44%. Data Mvmnt is at 285 microseconds or 26%. Of the data movement category, device copy easily dominates at 196 microseconds or 18%. In the Protl category, UDP Protl is very small; most of the contribution is from the device driver, IP, and the link layer. Syscall Argchk is large, since, as mentioned earlier, the receive buffer of the user-level UDP workload generator is larger. The Mbuf category is considerably more lopsided than expected; Mbuf Alloc is very large, reflecting a large number of small mbufs, yet Mbuf Misc is smaller than it is for any message length greater than or equal to 204 bytes.

3.5 Throughputs

In [8], Clark et al. make use of a particularly interesting means of analysis. They discuss the performance of various components of network software in terms of throughput. We present a similar analysis here, except that we include message size as a parameter. Throughput of a given operation in megabytes/second for a particular message size is calculated by dividing the message size by the time of each operation in microseconds. That definition has the side effect that an operation which consumes an amount of time approximately independent of message size will show rising throughput with message size. Figures 14a and 14b show the send and receive side throughputs for TCP. The Error Check, Data Structs, and OS categories are each capable of supporting far higher throughputs than the Mbuf, Data Mvmnt, Protls, and Checksum categories, so Figures 15a and 15b are show TCP throughputs just for those bottleneck throughput categories. Figures 16a and 16b show the send and receive side bottleneck throughputs for small TCP messages.

.

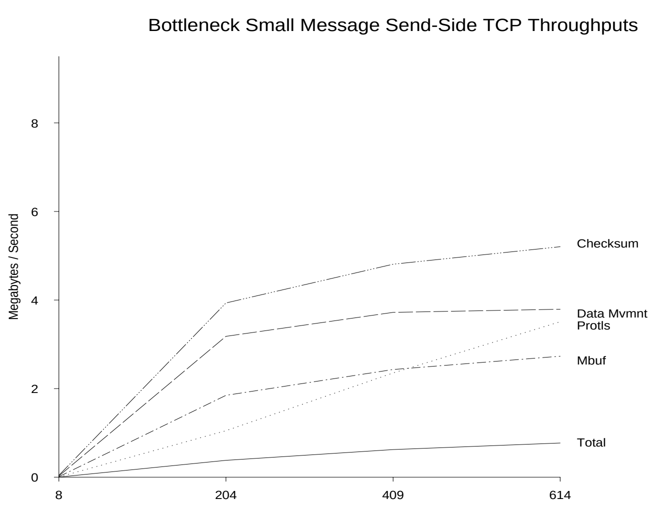

Figure 16a

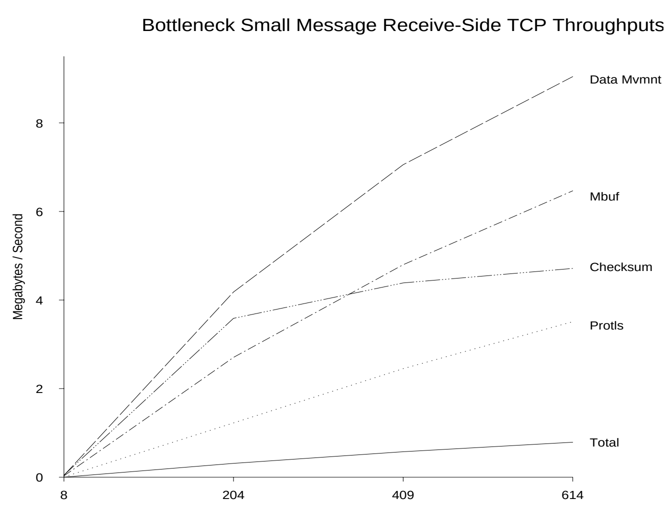

Figure 16b

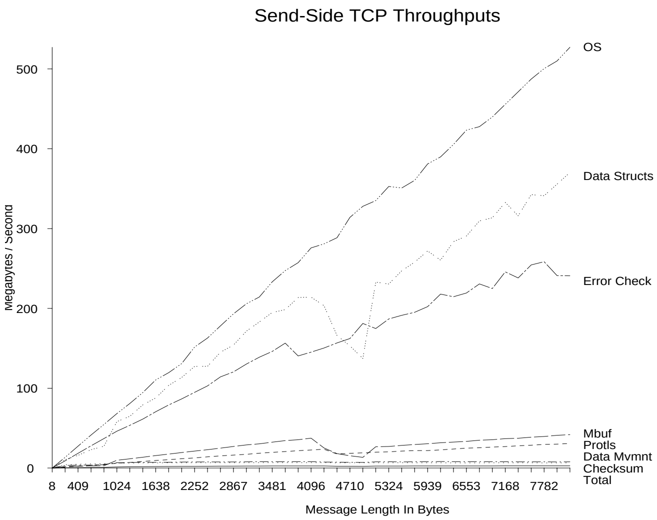

Figure 14a

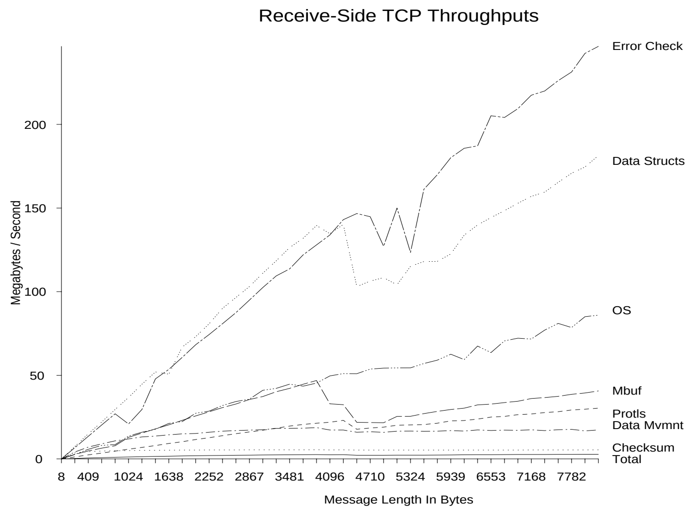

Figure 14b

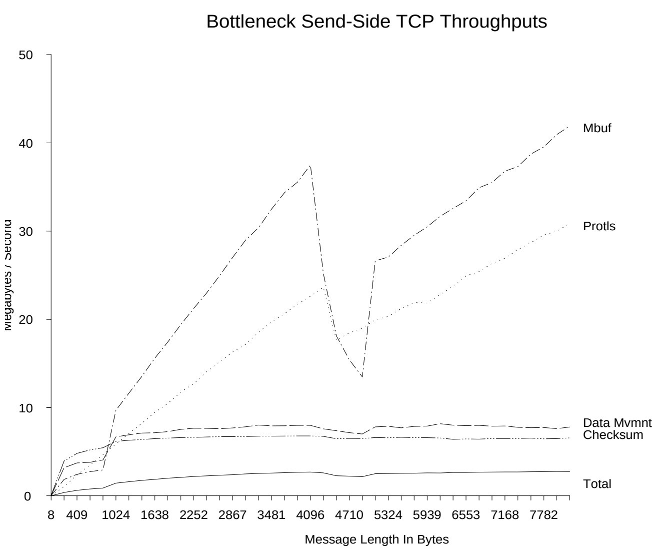

Figure15a

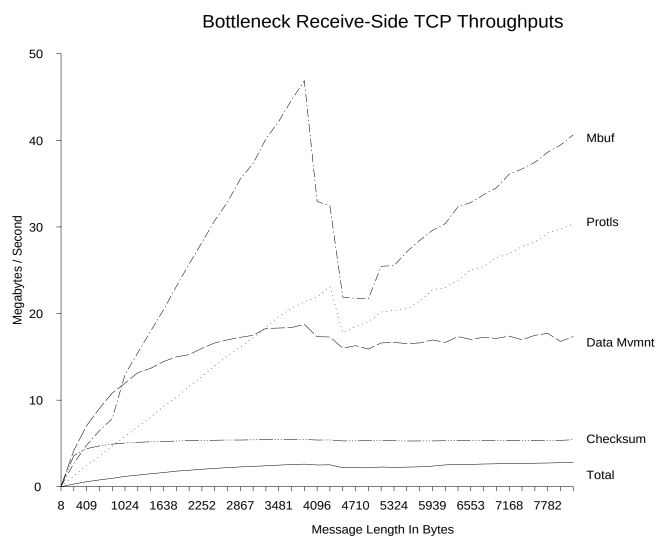

Figure 15b

For large messages, the Total curve, which is the actual DECstation TCP throughput, flattens out at approximately 2.5 MB/s (twice Ethernet bandwidth, 1/5 FDDI bandwidth) for both sender and receiver. The maximal throughput achieved is 2.75 MB/s, for 8192-byte messages. Interestingly, a size of 3891 bytes, at 2.61 MB/s, would prove to be a marginally better candidate for an FDDI MTU than the actual MTU - the throughput for 4300-byte messages is 2.52 MB/s. That effect and the “step” just below 1024 bytes, are due to the mbuf allocation strategy; a large dip in mbuf throughput can be seen there. At 1433 bytes, close to Ethernet MTU, the throughput is down to 1.49 MB/s. The major throughput bottlenecks, as was visible in the earlier graphs, are checksumming and data movement. If those bottlenecks could be somehow eliminated, there would be no major bottlenecks until about the FDDI throughput level of 12.5 megabytes/sec. On the other hand, reaching true gigabit throughputs on this architecture would require elimination of all categories save Data Structs and Error Check; even though the I/O bus is theoretically capable of 0.8 gigabits/sec [9], use of that amount of throughput by TCP/IP running on a DECstation 5000/200 does not seem likely.

The curves look rather different for small messages. The throughput for messages of size 614 is 0.773 MB/s, not quite 2/3 Ethernet throughput, about 1/6 the figure at the FDDI MTU. Fortunately, the MTU Discovery effort [15] would allow transmission of maximal sized messages over wide area networks; in light of the factor of six performance hit, such work could have a large effect on throughput of connections over T3 links. Here the Protls category is actually the worst bottleneck, and Data Mvmt, closely followed by Checksum, is the least important bottleneck.

3.6 An Analytic Model

The above understanding of performance factors can be used to design a simple but relatively accurate analytic model for processing time as a function of message size. It is of the form

$$t = A\ us + B\ us/byte * s + C\ us/mbuf * b + D\ us/frag * f \qquad \text{(EQ 1)}$$

T is the processing time, A is a constant number is microseconds (*us*), s is the message size in bytes, B is a constant number of *us/byte*, b is the number of small mbufs allocated by the sending side of the mbuf allocation algorithm, C is a constant number of *us/mbuf*, f is the number of extra fragments needed to send a message (f is 0 for a 1234-byte message and 1 for a 6543-byte message), and D is a constant number of *us/frag*. Since b and f are both functions of s, T is really a function of s. The reasons behind this choice of models can be seen in many of the graphs, especially Figures 3 and 4. There are two main regions in the graph, both with the same slope but offset at the FDDI MTU. There are also two humps in the regions where small mbufs are allocated, between 0 and 1024 bytes and between 4096 and 5120 bytes. It was clear early on in our analysis that both fragmentation and the number of small mbufs influence the processing time for a number of operations. We define messages less than FDDI MTU to contain zero fragments rather than one because the cost of the first fragment is higher the cost of the second for many operations.

3.6.1 TCP Fit

For comparison purposes, we fit both Equation 1 and a line to the total processing time curve for TCP in Figure 3 and computed coefficients of determination. The fit under Equation 1 is

$$t = 1040\ us + 0.494\ us/byte * s + 0.495\ us/mbuf * b + 705\ us/frag * f \qquad \text{(EQ 2)}$$

The linear least squares fit is

$$t = 885\ us + 0.612\ us/byte * s \qquad \text{(EQ 3)}$$

The coefficient of determination for Equation 2 is 99.7% (this study is only good to three digits of accuracy). The coefficient of determination for Equation 3 is 98.3%. The fit for the send and receive sides of total TCP processing are given, respectively, by

$$t = 445\ us + 0.264\ us/byte * s + 0.356\ us/mbuf * b + 324\ us/frag * f \quad \text{(EQ 4)}$$

$$t = 612\ us + 0.227\ us/byte * s + 0.142\ us/mbuf * b + 460\ us/frag * f \quad \text{(EQ 5)}$$

3.6.2 UDP Fit

The fit for total UDP processing time is

$$t = 913\ us + 0.493\ us/byte * s + 0.303\ us/mbuf * b + 749\ us/frag * f \quad \text{(EQ 6)}$$

and the fits for total UDP transmission and reception are

$$t = 477\ us + 0.264\ us/byte * s + 0.226\ us/mbuf * b + 421\ us/frag * f \quad \text{(EQ 7)}$$

$$t = 450\ us + 0.237\ us/byte * s + 0.081\ us/mbuf * b + 586\ us/frag * f \quad \text{(EQ 8)}$$

Equation 4 is similar to the TCP characterization, except that constant and per-mbuf costs are lower and fragmentation cost is somewhat higher. Given UDP's simplicity, the lower constant (UDP versus TCP Protl) and per mbuf costs are expected, but the higher fragmentation cost is not, as we discuss earlier.

3.6.3 Data-Touching Fits

Processing time for checksumming TCP messages is

$$t = 39.0\ us + 0.323\ us/byte * s + 0.0147\ us/mbuf * b + 93.8\ us/frag * f \quad \text{(EQ 9)}$$

This is very much the breakdown of a data-size-dependent operation. The us/byte coefficient forms most of the per-byte component of the total processing time curves, and the other coefficients are small.

The characterization for copying TCP-generated packets to the FDDI controller is

$$t = 29.2\ us + 0.086\ us/byte * s + 0.0412\ us/mbuf + 12.3\ us/frag \quad \text{(EQ 10)}$$

Already *us/byte* component is way down. There is a larger *us/mbuf* component because data starts out in mbufs, and thus this operation includes looping through a chain of mbufs, each with a different amount of data.

Time for copying TCP messages between user and kernel is given by

$$t = 37.7\ us + 0.061\ us/byte * s + 0.057\ us/mbuf * b + 75.2\ us/frag * f \quad \text{(EQ 11)}$$

The *us/byte* component is still further down.

3.6.4 "Constant" Cost Fits

We fitted the model to several of the more expensive operations that do not rise directly with message size. One was Mbuf Alloc, which resulting in a fit of:

$$t = 115\ us + 0.006\ us/byte * s + 0.180\ us/mbuf * b + 119\ us/frag * f \qquad (EQ\ 12)$$

Not surprisingly, the per-mbuf cost is high and per-byte cost almost nonexistent. Constant and fragmentation costs are approximately equal because both reflect allocation of a single cluster mbuf.

Another operation fit made was Mbuf Misc. We present the category rather than a single operation because the individual operations are rather small.

$$t = 57.4\ us + 0.141\ us/mbuf * b + 59\ us/frag * f \qquad (EQ\ 13)$$

The *us/byte* component does not appear because it is so small as to be within the statistical noise.

In addition, we fit the transport layer protocol costs. TCP Protl was fit by

$$t = 130\ us + 0.001\ us/byte * s - 0.011\ us/mbuf * b + 52\ us/frag * f \qquad (EQ\ 14)$$

The UDP Protl operation itself contributes very little to UDP processing time, but it is of some interest for comparison with TCP Protl, and for later comparison with earlier work. It is given by:

$$26.7\ us + 1\ us/frag * f \qquad (EQ\ 15)$$

4.0 Summary

We presented detailed measurements of various components of the TCP/IP and UDP/IP protocol stacks on a DECstation 5000/200 running Ultrix 4.2a. We showed that checksum computation and data transfers dominate component times for a real LAN workload. We concluded that using large packet MTUs (maximum transmission units) is very important to achieving high throughput, but that it will be difficult to improve latency. TCP and UDP time breakdowns are quite similar, suggesting that "lightweight" transport protocols are not likely to greatly decrease processing time. Finally, we presented analytical models for network software processing times.

5.0 References

[1] R. T. Braden, "Requirements for Internet Hosts -- Communication Layers," Internet RFC 1122, October 1989.

[2] R. T. Braden, D. A. Borman, and C. Partridge, "Computing the Internet Checksum," Internet RFC 1071, Sep. 1988.

[3] L.-F. Cabrera, E. Hunter, M. J. Karels, D. A. Mosher, “User-Process Communication Performance in Networks of Computers,” *IEEE Transactions on Software Engineering*, 14(1), 38-53, Jan. 1988.

[4] CCITT, “Data Communication Networks - Open System Interconnection (OSI) Model and Notation, Service Definition,” Blue Book Recommendations X.200 - X.219, International Telecommunications Union, 1989.

[5] K. C. Claffy, G. C. Polyzos, H. W. Braun, “Traffic Characteristics of the T1 NSFNET,” UCSD Technical Report CS92-252, July 1992.

[6] D. D. Clark, “Modularity and Efficiency in Protocol Implementation,” Internet RFC, 817, 1982

[7] D. D. Clark, “The Structuring of Systems Using Upcalls,” *Proceedings of the Tenth ACM Symposium on Operating System Principles*, 171-180, Dec. 1985

[8] D. D. Clark, V. Jacobson, J. Romkey, H. Salwen, “An Analysis of TCP Processing Overhead,” *IEEE Communications Magazine*, 23-29, June 1989

[9] Digital Equipment Corporation, TURBOchannel Hardware Specification, September 1991.

[10] R. Gusella, “A Characterization of the Variability of Packet Arrival Processes in Workstation Networks,” Technical Report UCB/CSD 90/612, UC Berkeley Computer Science Division, December 1990.

[11] V. Jacobson, “BSD TCP Ethernet Throughput”, comp.protocols.tcp-ip, Usenet, 1988

[12] V. Jacobson, R. Braden, D. Borman, “TCP Extensions for High Performance,” Internet RFC 1323, May 1992.

[13] S. J. Leffler, M. K. Mckusick, M. J. Karels, J. S. Quarterman, *The Design and Implementation of the 4.3 BSD UNIX Operating System*, Addison-Wesley, November 1989.

[14] J. C. Mogul, “Network Locality at the Scale of Processes,” *Proceedings of the SIGCOMM ‘91 Symposium on Communications Architectures and Protocols*, August 1991.

[15] J. C. Mogul, S. Deering, “Path MTU Discovery,” Internet RFC 1191, November 1990.

[16] S. Owicki, “Autonet Performance,” lecture at UC Berkeley, September 17, 1992.

[17] D. C. Plummer, “Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware,” Internet RFC 826, November 1982.

[18] J. B. Postel, “Transmission Control Protocol,” Internet RFC 793, September 1981.

[19] Sun Microsystems, Inc., “NFS: Network File System Protocol Specification,” Internet RFC 1094, March 1989.

[20] Sun Microsystems, Inc., “RPC: Remote Procedure Call specification: version 2,” Internet RFC 1057, June 1988.

[21] Sun Microsystems, Inc., “XDR: External Data Representation Standard,” Intern et RFC 1014, June 1987.

[22] R. W. Watson, S. A. Mamrak, "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices," *ACM Transactions on Computer Systems*, 5(2), 97-120, May 1987.

[23] J. Kay, J. Pasquale, "Measurement, Analysis, and Improvement of UDP/IP Throughput for the DECstation 5000," to appear, Proceedings of the Winter 1993 Usenix, January 1983.