The Importance of Non-Data Touching Processing Overheads in TCP/IP

Jonathan Kay and Joseph Pasquale

Computer Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093-0114

{jkay, pasquale}@cs.ucsd.edu

Abstract

We present detailed measurements of various processing overheads of the TCP/IP and UDP/IP protocol stacks on a DECstation 5000/200 running the Ultrix 4.2a operating system. These overheads include data-touching operations, such as the checksum computation and data movement, which are well known to be major time consumers. In this study, we also considered overheads due to non-data touching operations, such as network buffer manipulation, protocol-specific processing, operating system functions, data structure manipulations (other than network buffers), and error checking. We show that when one considers realistic message size distributions, where the majority of messages are small, the cumulative time consumed by the non-data touching overheads represents the majority of processing time. We assert that it will be difficult to significantly reduce the cumulative processing time due to non-data touching overheads.

1.0 Introduction

The goal of this study is to determine the relative importance of various processing overheads in network software, in particular, the TCP/IP and UDP/IP protocol stacks. In the past, significant focus has been placed on maximizing throughput, noting that "data touching" operations such as computing checksums and data copying are responsible for the primary bottlenecks in throughput performance [3, 5-8]. Maximal throughput is typically achieved by sending large messages. However, most TCP/IP packets on local area networks are smaller than 200 bytes [9]. Furthermore, almost all IP packets sent across wide-area networks are no more than 576 bytes long [4] because of the suggested default TCP maximum segment size [19]. The relative influences of processing overheads for packets whose sizes are at most 576 bytes greatly differ from those of packets whose lengths are the maximum transmission units (MTUs) for typical LANS such as Ethernet (1500 bytes) and especially FDDI (4352 bytes).

We present detailed measurements of processing times for various interesting categories of overhead in both TCP/IP and UDP/IP, for a wide variety of message sizes. More importantly, we present detailed aggregated times based on a realistic distribution of packet sizes generated by a measured workload from a real environment. As expected from studies of LAN packet traces, the overwhelming majority of messages are small and and do not require significant time in computing checksums or copying of data.

We show that when a realistic distribution of messages sizes is considered, although the checksum computation and data movement are by themselves large expenses, they contribute 40% (a minority) of the overall processing time. Consequently, we show that the cumulative effect of other overheads, such as protocol-specific processing and network buffer allocation, is important. This paper details which processing overheads are most expensive, and how the predominance of small messages affects the resulting breakdown of overheads in unexpected ways.

The paper is organized as follows. Section 2 describes our categorization of major network software processing overheads. Section 3 describes the experimental setup used to obtain measurements. Section 4 contains an analysis of network software processing overheads for a full range of message sizes. Section 5 presents a message size distribution obtained from measurements we carried out on a real workload. Section 6 analyzes aggregated processing overheads based on the measured message size distribution. Section 7 contains a finer analysis for each category of overheads, with emphasis on which individual overheads are most time-consuming. Finally, we present conclusions in Section 8.

2.0 Overhead Categories

We categorized the major processing overheads in network software as follows (the name abbreviations used in subsequent graphs are in parentheses): checksum computation ("Checksum"), data movement ("DataMove"), data structure manipulations ("Data Struct"), error checking ("ErrorChk"), network buffer management ("Mbuf"), operating system functions ("OpSys"), and protocol-specific processing ("ProtSpec"). Other studies have shown some of these overheads to be expensive [3, 5-8, 11, 22].

Checksum: Computing checksums is accomplished by a single procedure, the Internet checksum routine [1], which is performed on data in TCP and UDP, and on the header in IP.

DataMove: Data movement includes work involved in the moving of data from one place to another. These operations are further categorized as follows: copying data between user and kernel buffers (Usr-Krnl Cpy), copying data out to the FDDI controller (Device Copy), and a cache coherency protocol (Cache Clear).

DataStruct: Data structure manipulation comprises manipulations of various data structures other than mbufs (see below). These data structures are the socket buffer (Socket Buffer), IP defragmentation queue (Defrag Queue), and interface queue (Device Queue).

ErrorChk: Error checking is the category of checks for user

This work was supported in part by grants from the National Science Foundation, Digital Equipment Corporation and TRW in connection with the Sequoia 2000 Project, and NCR Corporation.

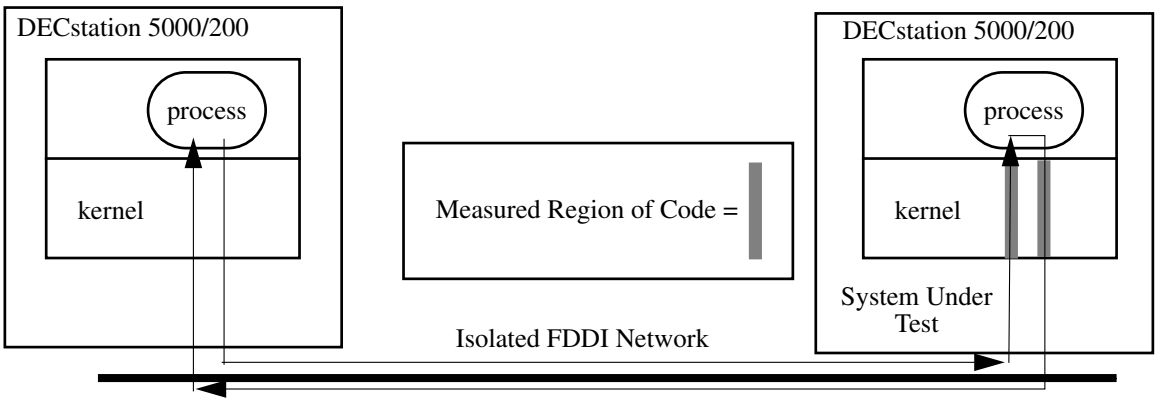


Figure 1: The experimental system consisted of two DECstation 5000/200 workstations connected by an FDDI network. A process on one machine sends a message to a process on the system under test, which then sends the same message back. The lines connecting the processes show the path of the message. Measurements are made on that part of executed code located in the kernel of the system under test, highlighted by the stipple-patterned lines.

and system errors, such as parameter checking on socket system calls.

Mbuf: All network software subsystems require a (sometimes complex) buffer descriptor which allows the prepending of headers and defragmentation of packets to be done cheaply. Berkeley Unix-based network subsystems buffer network data in a data structure called an *mbuf* [13]. All mbuf operations are part of this category. Allocation and freeing of mbufs are the most time-consuming mbuf operations.

OpSys: Operating system overhead includes support for sockets, synchronization overhead (sleep/wakeup), and other general operating system support functions.

ProtSpec: This category includes protocol-specific work such as setting header fields and maintaining state which is not included in any of the other categories. This category is more narrowly defined than what is often called "protocol processing". For example, although checksumming is usually considered part of TCP, UDP, and IP "protocol processing", we categorize checksumming separately because of its high expense, and because it is not limited specifically to any one of these protocols.

Other: This final category of overhead includes all the operations which are too small to measure. An example of this is the symmetric multiprocessing locking mechanism called frequently in DEC's Ultrix 4.2a, the operating system we measured. We estimate that it is called hundreds of times per message, but each call consumes close to the same amount of time as the overhead of executing our probes, and thus we are unable to tell with certainty how much time is consumed by that mechanism. In the processing times we present, the time due to "Other" is the difference between the total processing time and the sum of the times of the categories listed above.

3.0 Experimental Setup

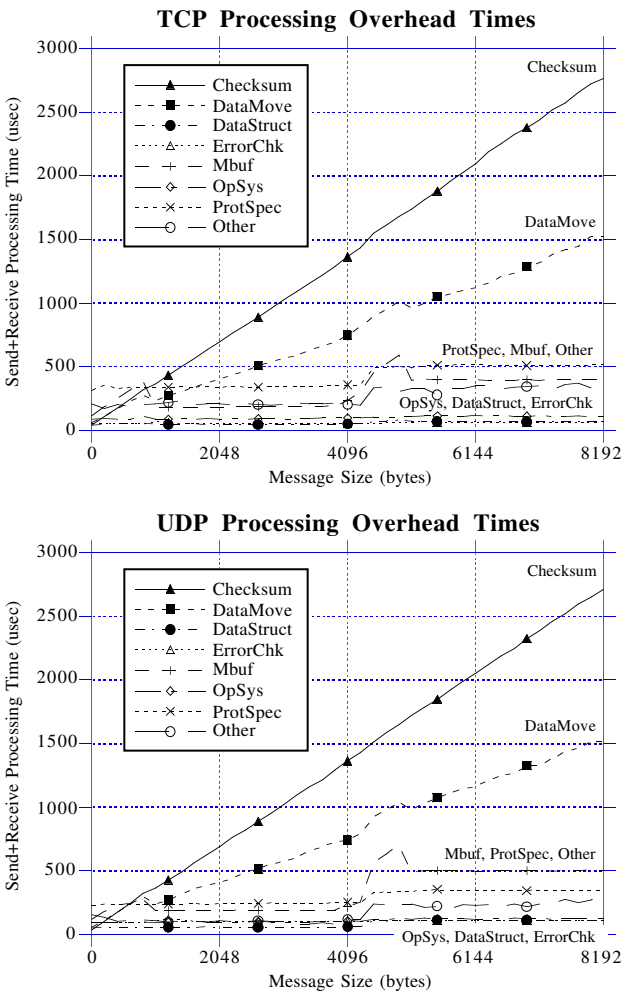
We instrumented the TCP/IP and UDP/IP protocol stacks in the DEC Ultrix 4.2a kernel. All measurements were taken on a DECstation 5000/200 workstation, a 19.5 SPECint MIPS RISC machine, connected to other similar workstations by an FDDI LAN. We connected an HP 1652B Logic Analyzer to the DECstation I/O bus to obtain software processing time measurements with a resolution of 40 nanoseconds (the DECstation clock cycle time). 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.

We measured the total message processing time for various components of network software when receiving and then sending the same-sized message. The experimental system, shown in Figure 1, 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. Each experiment is carried out for 40 message sizes evenly spaced from 1 byte to 8192 bytes. Experiments are repeated 100 times at the same message size to obtain statistical significance in the results (the average error was within $\pm 5\%$).

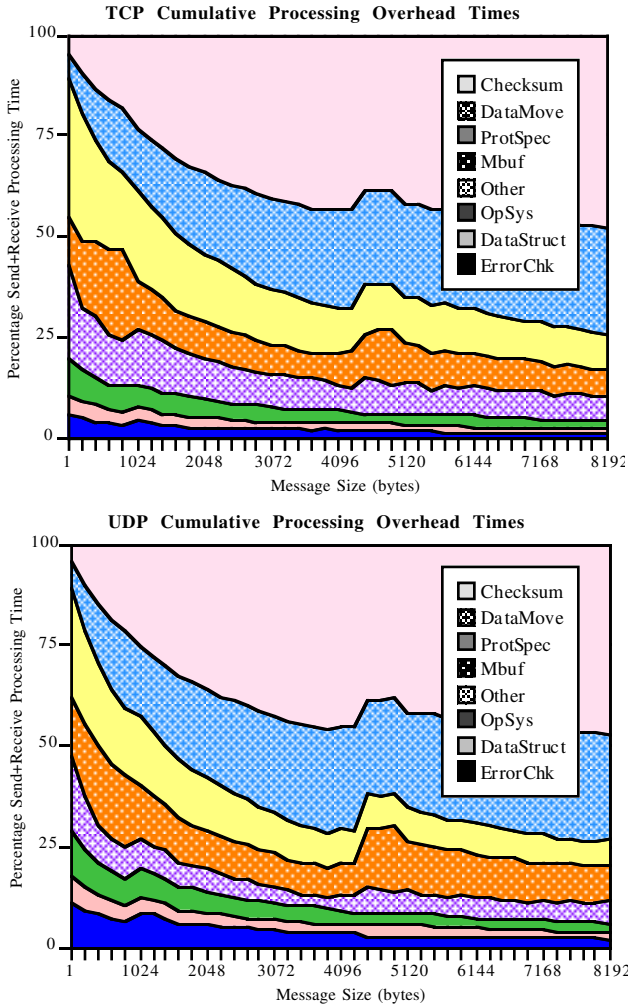
The experiments we designed are intended to specifically capture only CPU time spent in processing network packets. Other delays in network communications are ignored here. Our timings do not include network transmission times. Nor do we count time that a packet is held up by flow or congestion control, except for the processing time needed to make the decision not to hold up the packet; our workload does not provoke flow or congestion controls into operation. We note that the described workload design has the result that TCP acknowledgments (ACKs) are very nearly always piggybacked.

4.0 Overhead Times by Message Size

This section presents results on processing overhead times by message size. These results are not only interesing in themselves, but they provide a basis for understanding the performance effects given a network workload. Figures 2a-b show the per-message processing times versus message size for the various overheads for TCP and UDP messages, for a large range of message sizes, 1 to 8192 bytes. One can distinguish two different types of overheads: those due to *data-touching* operations (i.e. operations which actually touch message data) and those due to *non-data touching* operations. Generally, data-touching overhead times scale linearly with message size, whereas non-data touching overhead times are roughly constant. This is somewhat oversimplistic, as non-data touching overhead times may be indirectly affected by message size. For example, non-data touching operation times usually rise



Figures 2a-b: Breakdown of processing overhead times for TCP (2a) and UDP (2b) message sizes ranging from very small, 0 bytes, to very large, 8192.



Figures 3a-b: Breakdown of processing overhead times expressed as cumulative percentages of the total processing overhead time for TCP (3a) and UDP (3b) message sizes.

with the number of allocated mbufs or the number of fragments into which a message is fragmented.

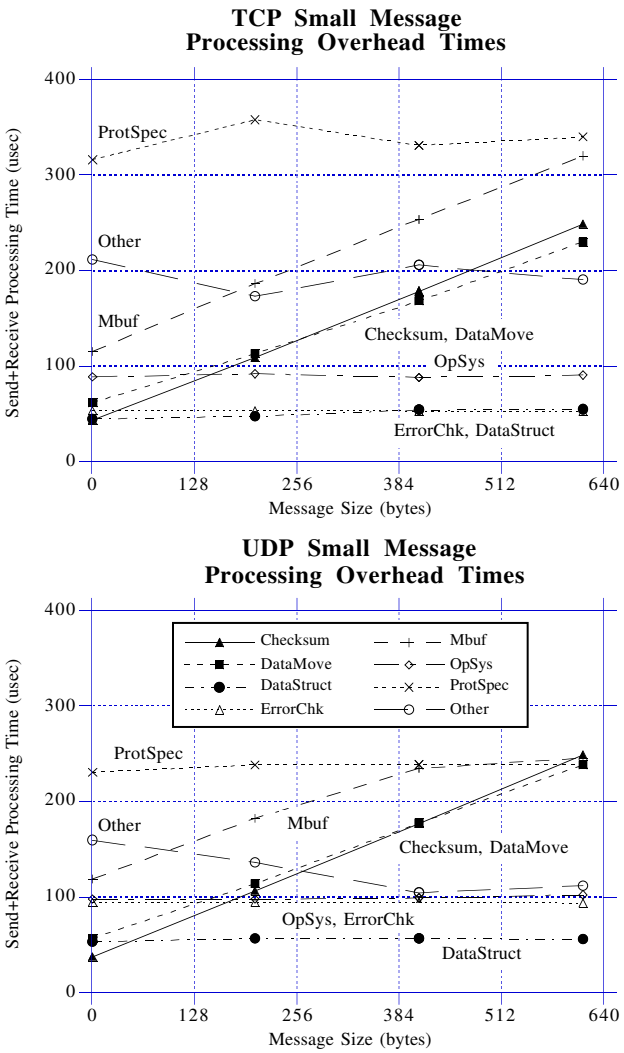
On the scales of message sizes presented in Figures 2a-b, the breakdowns of processing times between TCP and UDP are very similar. Even in the case of ProtSpec, TCP is only slightly more expensive than UDP. This is explained in that although TCP is the most complicated portion of the TCP/IP implementation, it is only a relatively small part of the executed layers of network software. Sending or receiving either a TCP or UDP message involves calling IP, the socket layer, the FDDI driver, and numerous support routines.

Another point, which is well-known, is that the processing overhead time for large messages is dominated by the data-touching overheads. Thus, it is clear that if one wishes to achieve high throughput by using maximally sized messages, the operations on which to focus attention are those that touch data.

An interesting feature of Figures 2a-b is the presence of various jumps and humps in the non-data touching overhead times. One reason for this is due to fragmentation: the MTU for FDDI is 4352 bytes, so TCP and IP fragment messages larger than 4312 bytes (FDDI MTU - sum of TCP and IP header lengths) into multiple packets, resulting in more operations.

Another reason for the humps is due to the design of the mbuf data structure. In the DEC Ultrix 4.2a network software implementation (as well as others based on Berkeley UNIX), an mbuf is a 128-byte structure. It can either directly store up to 108 bytes of data or indirectly point 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 connected into an ‘mbuf chain’ to contain larger packets, or to prepend protocol headers to mbuf chains. Mbufs use a complex allocation algorithm. To reduce internal fragmentation, the socket-send code allocates a string of up to 10 small mbufs if the message size is not greater than 1024 bytes, calling a memory allocation routine for each mbuf; this explains the humps below 1024 bytes. For messages larger than 1024 bytes but less than a page, a 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 not greater than 5120 bytes (i.e. 1024 bytes more than a 4096 byte page), whereas a second cluster mbuf is allocated for messages larger than 5120 bytes. Others have remarked on potential problems due to this bimodality [10], and the resulting bimodality is apparent in [3].

Figures 3a-b present the data differently, showing the breakdown of processing overhead times expressed as cumulative per-



Figures 4a-b: Breakdown of processing overhead times for small TCP (4a) and UDP (4b) message sizes.

centages of the total processing overhead time. Notice that the upper two regions in Figures 3a-b are due to the data-touching overheads; for large message sizes these operations consume approximately 70% of the total processing overhead time. However, as messages get smaller, the non-data touching overhead times expectedly become more prominent. In fact, for single byte messages, data-touching overheads contribute only 11% of the total processing overhead time.

In Figures 4a-b, we magnify the leftmost regions of the graphs in Figures 2a-b to focus on small messages. The message sizes range from 0 to 614, which includes the most common sizes of packets [4] sent over an Internet. For small messages, especially those smaller than 128 bytes which is generally enough for RPC's and acknowledgments, the non-data touching overheads clearly dominate.

5.0 TCP and UDP Message Size Distributions

To determine realistic TCP and UDP message size distributions, we obtained a packet trace from an FDDI network used primarily as a LAN for workstations and their file servers from a university computer science department, using the program `tcpdump`. 139,720 message sizes were collected; 90% of the messages are UDP messages, mostly generated by NFS. The rest are TCP messages. We defined "message size" to mean the amount of actual user data sent, not including any protocol headers. Message sizes were computed from the information contained in the packets. Figures 5a-b show the message size distributions for TCP and UDP messages.

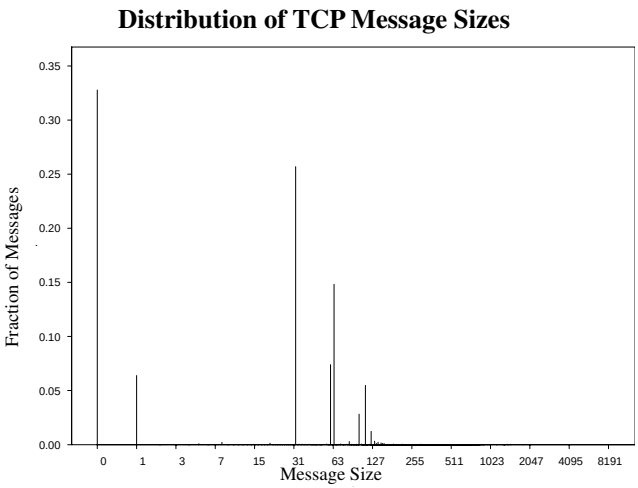


Figure 5a: Distribution of TCP message sizes. Almost all TCP messages are small (0.5% of the TCP messages are actually larger than 256 bytes - they are merely invisible in the graph).

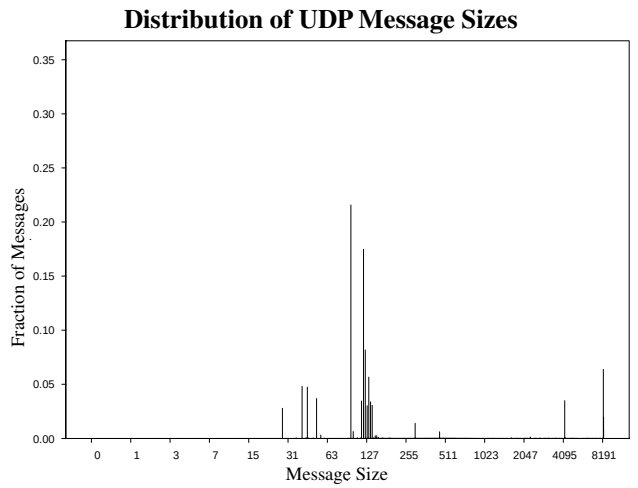


Figure 5b: Distribution of UDP message sizes. As in Figure 5a, most of the messages are small. However, a significant fraction of the messages are very large.

The observed message size distributions matched our expectations based on previous work on Ethernet-based packet traces [9, 15]. Figure 5a shows that almost all the TCP messages are small; over 99% of all the TCP messages are less than 200 bytes long. Figure 5b shows a bimodal distribution for UDP message sizes. The great majority of UDP messages are small, but there are some very large messages: 86% of the UDP messages are less than 200 bytes long, while 9% are at least 8192 bytes long.

The median message sizes for TCP and UDP messages are 32 and 128 bytes, respectively. The low UDP median reflects the fact, visible in the distributions, that even in the case of UDP, most messages are small (e.g. NFS status or directory reading messages). The reason for the large number of 8 kilobyte UDP messages is NFS. The scarcity of large TCP messages (there are a few large TCP messages, as large as 2048 bytes, just not enough to be visible

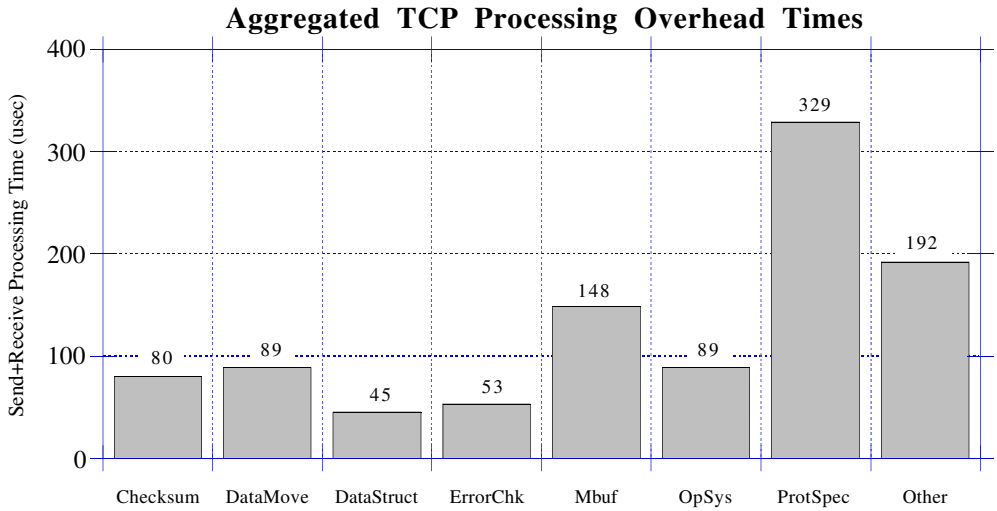


Figure 6a: Profile of aggregate times for TCP processing overheads. The most time-consuming overhead is protocol-specific processing, which is a non-data touching overhead. Since almost all TCP messages are small, the data-touching overheads consume a small amount of time.

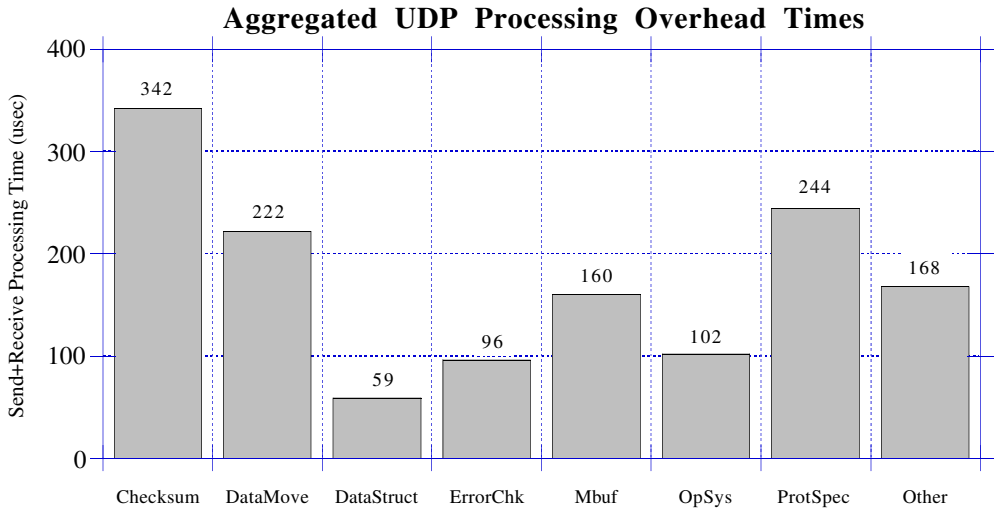


Figure 6b: Profile of aggregate times for UDP processing overheads. Since there are a significant number of large UDP messages, the data-touching overheads (Checksum and DataMove) are major time consumers. However, they still consume less total time than do the non-data touching overheads.

in Figures 5a-b) was surprising at first. As a check, we observed an Ethernet which serves as the LAN for general computing in a different university computer science department, to see if this behavior was simply an artifact of the particular FDDI network we measured, but we found very similar results. We determined that the most common application that produces large TCP messages is remote file distribution, which on workstations is done increasingly via NFS using UDP on most LANs. Thus, most of the large messages we observed in our environments are UDP messages. There are other applications that produce large TCP messages, such as X Window System image transfer and Network News Transfer Protocol, but these messages are relatively infrequent.

One would expect a detectable fraction of messages of size 536 (576 - size of headers), the TCP Default Maximum Segment Size, for "large" wide-area network data packets [4, 19]. However, there is no indication of this. Our data actually contained packets of this size, but the fraction is too small to show up in the graphs. Thus, most large packets we observed are local.

6.0 Aggregate Overhead Times

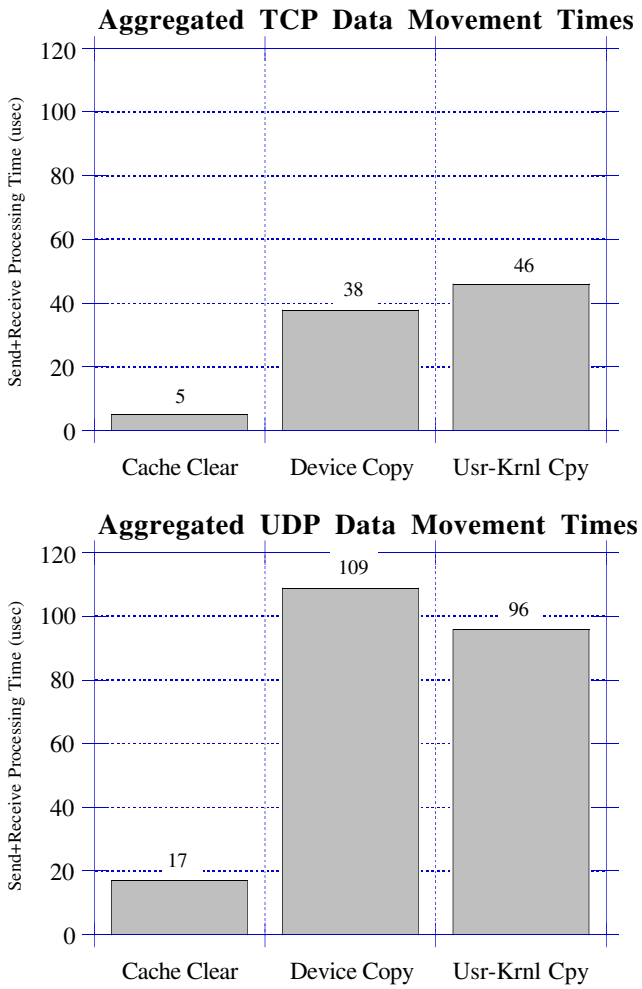
We have established that network software processing overhead times for small messages depend mainly on non-data touching operations, while for large messages the processing overhead times depend mainly on data-touching operations. We have also established that the distributions of TCP and UDP message sizes can be very different in a real workload. Thus, the categories of processing overheads we defined in Section 2 have different relative costs for TCP and UDP because of the differing distributions of message sizes. We now consider the aggregate processing overhead times based on the message size distributions given in Figures 5a-b. Figures 6a-b show aggregate processing overhead times for TCP and UDP.

6.1 The Importance of Non-Data Touching Overheads

Figure 6a shows that only a small fraction (16%) of the total time spent processing TCP messages is due to checksumming and

data movement. In the case of UDP shown by Figure 6b, checksumming and data movement are significant factors, but they do not overwhelm the other categories of overhead, and they do not represent the majority of the total processing time. Consequently, the non-data touching operations have a major effect on performance.

In considering reasons for the large size of the non-data touching overheads, we do not believe that the Berkeley Unix network subsystem is poorly tuned. Our observation is that the Berkeley Unix implementation of the TCP/IP protocol suite has very rich functionality. Even the UDP/IP protocol stack under Unix has a great deal of functionality, despite UDP's reputation as a minimal protocol: it has its own port space, checksum integrity, scalable internetworking, fragmentation, dynamic LAN address resolution, two-level input buffering and queuing, an error-checked system call interface to user programs, etc. Most of this functionality is due to layers other than UDP. Each bit of functionality carries a cost, the sum of which is significant. Consider that the TCP portion of the ProtSpec (i.e. protocol-specific processing) category accounts for only 13% of the total processing time for single-byte messages (whose processing times are least affected by data-touching operations); all the other time is consumed by functionality provided by other layers and by the operating system.



Figures 7a-b: Aggregate data movement times for TCP (7a) and UDP (7b) messages. Copying times are much higher for UDP because of the greater number of large UDP messages.

6.2 Difficulty of Optimizing Non-Data Touching Overheads

Eliminating the single operation of checksumming is sufficient to produce a large reduction in processing time for large messages. An increasing number of TCP/IP implementations succeed in reducing or avoiding the checksumming operation by various strategies. Some vendors have disabled UDP checksumming by default at a loss in packet integrity and in contravention of Internet host requirements [2]. Clark and Tennenhouse have suggested combining checksumming and copying using integrated layer processing [8]. Silicon Graphics, Inc. workstations move the burden of performing the checksum computation from the CPU to their FDDI interfaces [20]. In [12], we have proposed a scheme intended to preserve almost all of the performance gain from disabling checksums with nearly none of the loss of integrity nor any additional cost in hardware. We suggest avoiding the TCP and UDP data checksum computations when the source and destination are on the same LAN and that LAN supports a hardware CRC. Since the overwhelming majority of packets stay within a single LAN, and those packets are covered by the LAN CRC, our checksum redundancy avoidance algorithm provides a dramatic performance improvement for most packets without loss of checksum protection. Such strategies can result in near-doubling of maximum throughputs.

Improving overall performance by optimizing non-data touching operations is more difficult. For example, because of the lack of large TCP messages, the most prominent category in the TCP profile in Figure 6a is ProtSpec. ProtSpec is a category made up of the protocol-specific processing overhead from each protocol layer (TCP, IP, Link layer, and FDDI driver). The largest component of ProtSpec is that of TCP, which as mentioned above consumes 13% of the total processing time; however, TCP protocol-specific processing is actually made up of a large number of smaller operations. The other prominent categories, Mbuf and Other, are similarly combinations of a number of smaller operations. Thus, a wide range of improvements would be needed to produce a significant improvement in performance for the non-data touching operations.

7.0 In-Depth Analysis

Section 6 presented an overview of the analysis of aggregate processing overhead times. This section takes a closer look at the aggregate times for individual categories, in decreasing order of importance. Each category is examined in detail and the most time-consuming factors in each category are explained. Differences between TCP and UDP time breakdowns are also explained. In general, the differences either result from differences between the distributions of message sizes or differences in TCP and UDP protocol complexity.

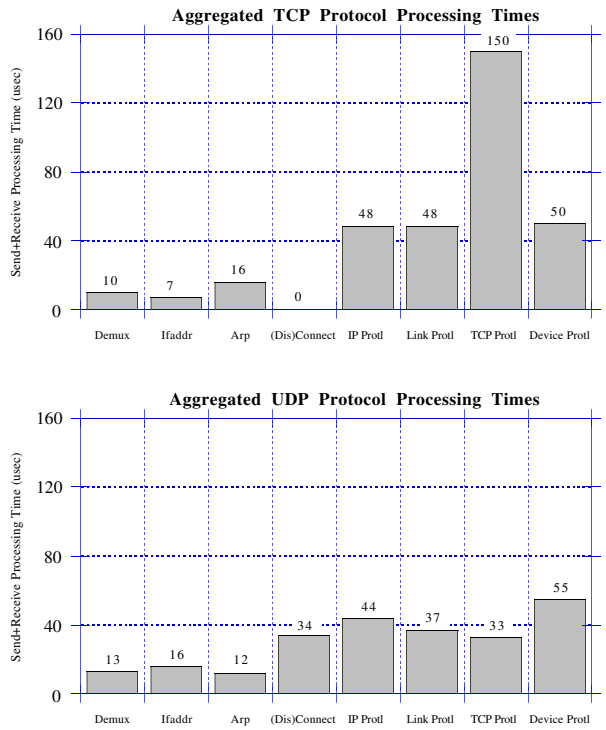
7.1 Touching Data

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

As was seen in Figures 2a-b and 3a-b, Checksum is the dominant overhead for large messages. From Figures 6a-b, we see that the aggregate times for Checksum are 48 usec and 356 usec for TCP and UDP message size distributions respectively. Since Checksum is itself a single operation, we do not analyze it any further.

DataMove is the next largest time-consuming category of overheads for large messages; see Figures 7a-b for a breakdown of this category. It includes three operations: Usr-Krnrl Cpy, Device Copy, and Cache Clear. Usr-Krnl Cpy is the amount of time spent copying incoming messages from kernel to user buffers and outgoing messages from user to kernel buffers. Device Copy and Cache

Clear concern movement of data between kernel buffers and the FDDI controller. The controller we had access to does not support send-side DMA, so the CPU must copy data to the controller. Device Copy is the amount of time needed to copy each message from kernel buffers to buffers in the FDDI controller. The controller does support receive-side DMA, so there is no receive-side equivalent to Device Copy. However, a cache coherency protocol is needed; Cache Clear is the amount of time used to insure cache consistency when the FDDI controller DMAs incoming packets into memory.



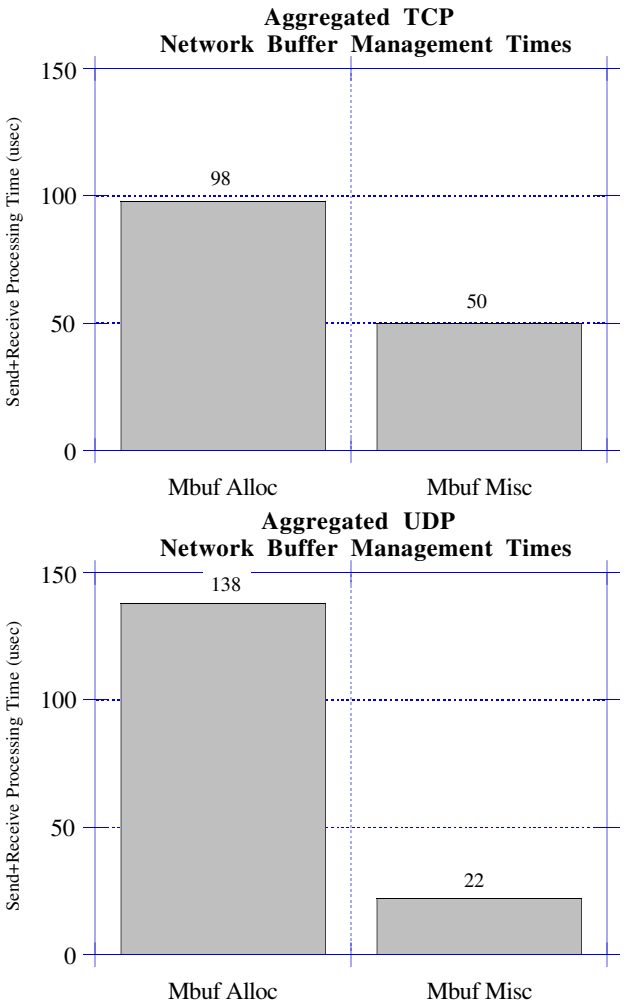
Figures 8a-b: Aggregate protocol-specific processing (ProtSpec) times under TCP (8a) and UDP (8b) messages. TCP protocol-specific processing time is large, but less than half of total protocol-specific processing time. In comparison, the UDP protocol-specific processing time is relatively small.

Time to copy data between the device and memory dominates the other times, with user-kernel copying time coming in a close second. In fact, Device Copy and Krnl Cpy do roughly the same amount of work, but caching improves the latter's time.

Since the time consumed by DataMove overheads increase with message size, the higher times for UDP reflect the higher average UDP message length. Another smaller source of difference between TCP and UDP is that Device Copy is lower than Usr-Krnl Copy in TCP, but the pattern is reversed for UDP. That is because each of the data-touching overhead times effectively has a constant component and a component rising linearly with message size. Usr-Krnl Copy has a larger constant component but a smaller linear component than Device Copy, and thus the reversal in relative sizes is due to the different TCP and UDP message size distributions.

7.2 Protocol-Specific Processing

As seen in Figures 6a-b, ProtSpec is the dominant category for TCP and is prominent for UDP. For TCP messages, ProtSpec consumes nearly half the total processing overhead time.

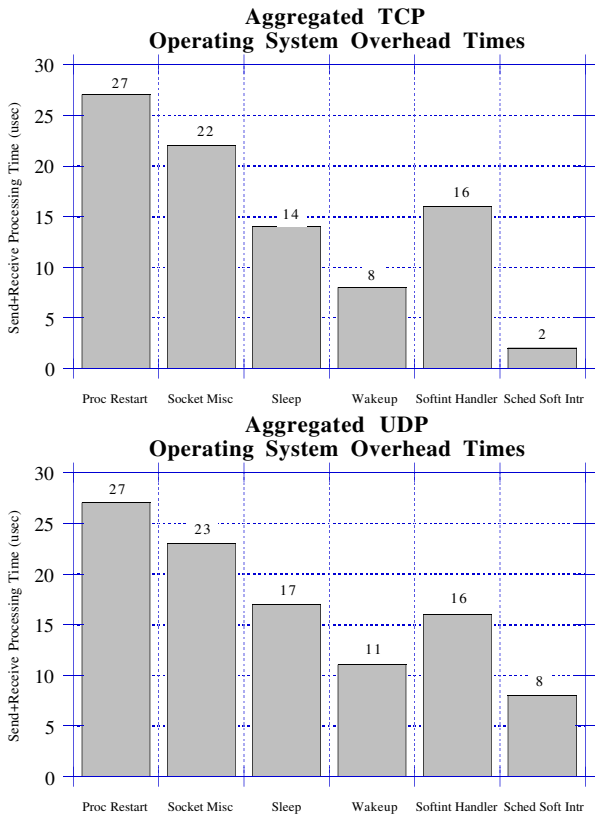


Figures 9a-b: Aggregate network buffer (i.e. "mbuf") management times for TCP (9a) and UDP (9b) message sizes. The large cost of the several memory allocation operations performed by the mbuf software is one of the largest single operation costs.

A breakdown of the ProtSpec category is shown in Figures 8a-b. Interface Protl is in the device driver layer, Link Protl is part of the IEEE 802 encapsulation layer, IP Protl is the IP layer, and TCP Protl is the TCP layer. Arp is the entire Address Resolution Protocol [17], Demux is the operation of finding a protocol control block, given a TCP or UDP 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).

TCP Protl dominates the ProtSpec category. This is in stark contrast to UDP, of which UDP Protl is a rather small portion. However, despite TCP Protl's size, it only consumes 13% of the total TCP/IP processing time. Reduction of TCP protocol-specific processing time would be useful, but it is questionable whether the performance improvement would be worth the major effort required. However, substantial reductions in the entire protocol-specific processing category can produce a significant performance improvement, as the category as a whole consumes 32% within TCP and 18% within UDP. Such a reduction would require significant improvements across the entire stack of protocols.

At first glance, it may be surprising that Demux is so small, under 20 microseconds, given reports that this operation is a bottleneck [7, 14]. This is because our single-user testing environment only has a single connection, and hence a single entry in the list of protocol control blocks.



Figures 10a-b: Aggregate operating system overhead times for TCP (10a) and UDP (10b) message sizes. Transfer of control is surprisingly inexpensive.

7.3 Mbufs

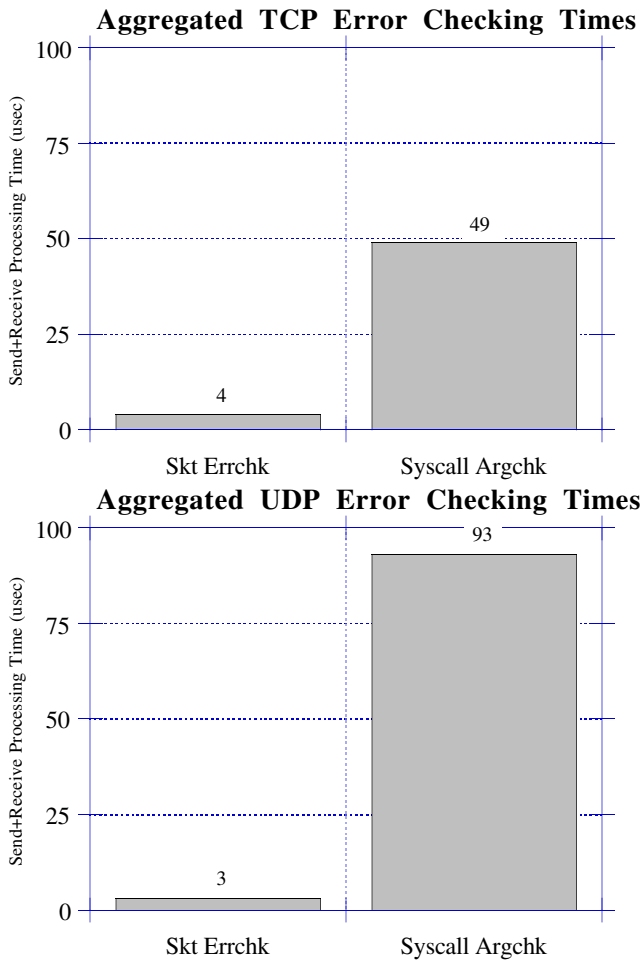
Mbuf is the second largest of the non-data touching categories of overhead. The mbuf data structure supports a number of operations, the most costly of which is Mbuf Alloc. Figures 9a-b contain breakdowns of the Mbuf category into mbuf allocation and deallocation (Mbuf Alloc) and all other mbuf operations (Mbuf Misc).

Mbuf Alloc is more expensive for two reasons: memory allocation is an inherently expensive operation and memory allocation is performed a number of times per message. Mbuf Alloc consumes more time for UDP because of the mbuf allocation strategy. Messages less than 1024 bytes long are stored in as many small buffers as are needed, each individually allocated to store at most 108 bytes. The overwhelming majority of TCP messages fit within a single mbuf, while many UDP messages are long enough to require two mbufs for data storage.

The other mbuf operations constituting Mbuf Misc are simpler operations such as copying mbuf chains (done to a certain extent with pointers), defragmentation (implemented by joining two linked lists of mbufs), and message length checking. TCP spends more time in Mbuf Misc because it must make a copy of each message in case retransmission is necessary.

7.4 Operating System Overheads

Figures 10a-b show the breakdown of operating system overheads. 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 a process. Proc Restart is the time needed for the sleeping process to start running again. Sched Soft Intr is the operation of scheduling a software interrupt to process the incoming packet, and Softint Handler is the software interrupt handler, 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 relative to the other categories of overheads.



Figures 11a-b: Aggregate error checking times for TCP (11a) and UDP (11b) message sizes. Checking that a user has specified a correct receive buffer is expensive and imposes a large penalty on large receive buffers.

7.5 Error Checking

ErrorChk is the category of checks for user and system errors. Figures 11a-b show the breakdown of overheads for this category. Skt Errchk contains 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 relatively large, especially for UDP. This is mostly because of time spent verifying that a user's buffer is of the correct size.

7.6 Data Structure Manipulations

The DataStruct category consists of manipulations of various data structures that are not so expensive as to be worth individual scrutiny as was the case with mbufs. Figures 12a-b show the break-

down of overheads for this category. The socket buffer is the data structure in which a limited amount of data is enqueued either for or by the transport protocol. TCP makes heavier use of this data structure. UDP makes no use of the socket buffer structure for outgoing messages, and only uses it as a finite-length queue upon reception. In contrast, TCP uses the socket buffer to implement windowing flow control on both send and receive sides.

The Defrag Queue is the data structure used to defragment IP packets. The times for Defrag Queue processing require explanation. In general, messages larger than the FDDI MTU must be sent in multiple pieces (fragmented and defragmented). Fragmentation is implemented in both IP and TCP. UDP messages are fragmented by IP, but TCP does its own fragmentation specifically to avoid IP fragmentation. Thus, it is surprising that the code checking the defragmentation queue is called at all for TCP messages; this reflects a place in the IP code where a check for matching fragments could have been avoided. Even more interesting is the fact that the overall amount of time spent in Defrag Queue for UDP is not noticeably greater than the minimal value. Less than 10% of UDP messages are large enough to be fragmented; the processing times of these large messages are not sufficient to noticably raise the average cost of defragmentation. This also implies that most of the cost of Defrag Queue derives from checking the defragmentation queue even when not necessary.

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 incoming data is enqueued by the device driver until IP is ready to process it by a software interrupt. UDP messages spend more time in Device Queue processing because of fragmentation. If a UDP message is fragmented into two FDDI frames, then twice as much work must be performed by the Device Queue to send that message.

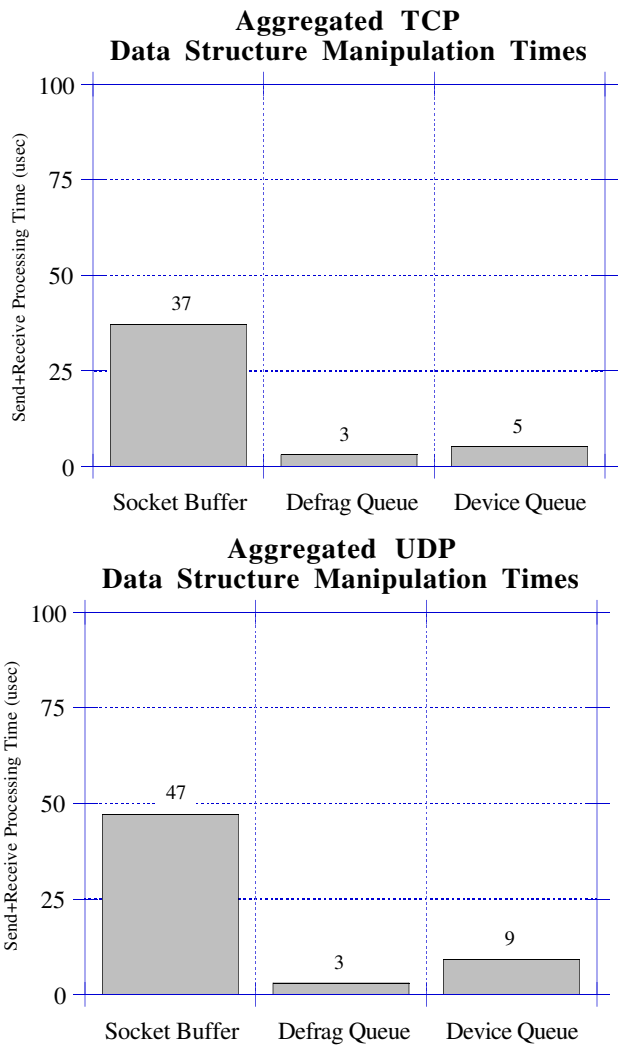
8.0 Conclusions

We presented detailed measurements of various categories of processing overhead times of the TCP/IP and UDP/IP protocol stacks on a DECstation 5000/200. We have shown that it is important to consider the effects of non-data touching overheads on performance, particularly because most messages observed in real networks are small. While others have appropriately focused on the significant costs in processing time of the data-touching operations, specifically computing checksums and data copying, these costs are most important when considering large message sizes. However, when one considers aggregate costs based on a realistic distribution of message sizes, the non-data touching overheads consume a majority of the total software processing time (84% for TCP, 60% for UDP).

Non-data touching overheads have received relatively little attention compared to data-touching overheads. There are significant ongoing efforts to reduce data-touching overheads (e.g. integrated layer processing, computing checksums in hardware, avoiding checksum computations redundant with hardware, and by restructuring operating system software to minimize data movement). Such efforts are effective because optimizing a single data-touching operation produces a large improvement in performance. Unfortunately, time is more evenly spread among the non-data touching overheads. Reducing a single non-data touching overhead, such as TCP protocol-specific processing, does not have a relatively significant effect on overall performance. Thus, a wide range of optimizations to non-data touching operations would be needed to produce a significant performance improvement.

9.0 Acknowledgements

We gratefully acknowledge the help of Keith Muller for helping with both the design and the debugging of the experiments described in this paper.



Figures 12a-b: Aggregate data structure manipulation times for TCP (12a) and UDP (12b) message sizes. TCP makes more extensive use of the socket buffer's properties than UDP.

10.0 References

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

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

[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, January 1988.

[4] R. Caceres, P. B. Danzig, S. Jamin, D. J. Mitzel. "Characteristics of Wide-Area TCP/IP Conversations," *Proceedings of the SIGCOMM '91 Symposium on Communications Architectures and Protocols*, pp. 101-112, August 1991.

[5] D. D. Clark, "Modularity and Efficiency in Protocol Implementation," Internet RFC, 817, 1982.

[6] D. D. Clark, "The Structuring of Systems Using Upcalls," *Proceedings of the Tenth ACM Symposium on Operating System Principles*, pp. 171-180, December 1985.

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

[8] D. D. Clark, D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," *Proceedings of the SIGCOMM '90 Symposium on Communications Architectures and Protocols*, pp. 200-208, September 1990.

[9] R. Gusella, "A Measurement Study of Diskless Workstation Traffic on an Ethernet," *IEEE Transactions on Communications*, 38(9), pp. 1557-1568, September 1990.

[10] N. C. Hutchinson, S. Mishra, L. L. Peterson, V. T. Thomas, "Tools for Implementing Network Protocols," *Software - Practice and Experience*, pp. 895-916, September 1989.

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

[12] J. Kay, J. Pasquale, "Measurement, Analysis, and Improvement of UDP/IP Throughput for the DECstation 5000," *Proceedings of the Winter 1993 USENIX Conference*, pp. 249-258, January 1993.

[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] P. E. McKenney, K. F. Dove, "Efficient Demultiplexing of Incoming TCP Packets," *Proceedings of the SIGCOMM '92 Symposium on Communications Architectures and Protocols*, pp. 269-279, August 1992.

[15] Mogul, J., "Network Locality at the Scale of Processes," *Proceedings of the SIGCOMM '91 Symposium on Communications Architectures and Protocols*, pp. 273-284, August 1991.

[16] J. Ousterhout, "Why Aren't Operating Systems Getting Faster As Fast As Hardware," *Proceedings of the Summer 1990 USENIX Conference*, pp. 247-256, June 1990.

[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. Postel, "Transmission Control Protocol," Internet RFC 793, September 1981.

[19] J. Postel, "The TCP Maximum Segment Size and Related Topics", Internet RFC 879, November 1983.

[20] V. J. Schryver, private communication, November 1992.

[21] Sun Microsystems, Inc., "NFS: Network File System Protocol Specification," Internet RFC 1094, March 1989.

[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), pp. 97-120, May 1987.