High-performance TCP/IP and UDP/IP Networking in DEC OSF/1 for Alpha AXP

1 Abstract

The combination of the Alpha AXP workstations, the DEC FDDIcontroller /TURBOchannel network interface, the DEC OSF/1 AXP operating system, and a streamlined implementation of the TCP/IP and UDP/IP delivers to user applications almost the full FDDI bandwidth of 100 Mb/s. This combination eliminates the network I/O bottleneck for distributed systems. The TCP /IP implementation includes extensions to TCP such as support for large transport windows for higher performance. This is particularly desirable for higher-speed networks and/or large delay networks. The DEC FDDIcontroller/TURBOchannel network interface delivers full bandwidth to the system using DMA, and it supports the patented point-to-point, full-duplex FDDI mode. Measurement results show UDP performance is comparable to TCP. Unlike typical BSD-derived systems, the UDP receive throughput to user applications is also maintained at high load.

We have seen significant increases in the bandwidth available for computer communication networks in the recent past. Commercially available local area networks (LANs) operate at 100 megabits per second (Mb/s), and research networks are running at greater than 1 gigabit per second (Gb /s). Processor speeds have also seen dramatic increases at the same time. The ultimate throughput delivered to the user application, however, has not increased as rapidly. This has led researchers to say that network I/O at the end system is the next bottleneck.[1]

One reason that network I/O to the application has not scaled up as rapidly as communication link bandwidth or CPU processing speeds is that memory bandwidth has not scaled up as rapidly even though memory costs have fallen dramatically. Network I/O involves operations that are memory intensive due to data movement and error checking. Scaling up memory bandwidth, by making memory either wider or faster, is expensive. The result has been an increased focus on the design and implementation of higher-performance network interfaces, the re-examination of the implementation of network I/O, and the consideration of alternative network protocols to achieve higher performance.[2,3,4].

This paper describes the work we did to remove the end system network I/O bottleneck for current commercially available high-speed data links, such as the fiber distributed data interface (FDDI).[5,6] We used the conventional internet protocol suite of transmission control protocol /internet protocol (TCP/IP) and the user datagram protocol/internet protocol (UDP/IP) on Alpha AXP hardware and software platforms.[7,8,9] The specific hardware platform was the DEC 3000 AXP Model 500 workstation with the DEC

FDDIcontroller/TURBOchannel adapter (DEFTA). The software platform was the DEC OSF/1 operating system version 1.2 using the TCP and UDP transport protocols. The combination of the Alpha AXP workstations, the DEFTA

adapter, the DEC OSF/1 operating system, and a streamlined implementation of the TCP/IP and UDP/IP delivers to user applications essentially the full FDDI bandwidth of 100 Mb/s.

While the DEC FDDIcontroller/TURBOchannel network interface is lower cost than previous FDDI controllers, it also delivers full bandwidth to the system using direct memory access (DMA). In addition, it supports the patented point-to-point, full-duplex FDDI mode. This allows a link to be used with 100 Mb/s in each direction simultaneously, which increases throughput in some cases and reduces latency compared to the standard FDDI ring mode.

Incremental work for data movement and checksums has been optimized to take advantage of the Alpha AXP workstation architecture, including 64-bit support, wider cache lines, and the coherence of cache blocks with DMA. Included in the TCP/IP implementation are extensions to TCP recently recommended by the Internet Engineering Task Force (IETF), such as support for large transport windows for higher performance.[10] This is particularly desirable for high-speed networks and/or large delay networks.

We feel that good overload behavior is also important. Workstations as well as hosts acting as servers see substantial load due to network I/O. Typical implementations of UDP/IP in systems based on the UNIX operating system are prone to degradation in throughput delivered to the application as the received load of traffic to the system increases beyond its capacity. Even when transmitting UDP/IP packets from a peer transmitter with similar capabilities, the receiver experiences considerable packet loss. In some cases, systems reach receive "livelock," a situation in which a station is only involved in processing interrupts for received packets or only partially processing received packets without making forward progress in delivering packets to the user application.[11] Changes to the implementation of UDP/IP and algorithms incorporated in the DEFTA device driver remove this type of congestion loss at the end system under heavy receive load. These changes also eliminate unfairness in allocation of processing resources, which results in starvation (e.g., starving the transmit path of resources).

The next section of this paper discusses the characteristics of the Alpha AXP workstations, the DEC OSF/1 operating system, and the two primary transport protocols in the internet protocol suite, TCP and UDP. We provide an overview of the implementation of network I/O in a typical UNIX system using the Berkeley Software Distribution (BSD) to motivate several of the implementation enhancements described in the paper.[12]

The section on Performance Enhancements and Measurements Results then describes the specific implementation enhancements incorporated in the DEC OSF/1 operating system version 1.2 to improve the performance of

TCP and UDP. This section also provides measurement results for TCP and UDP with DEC 3000 AXP workstations running DEC OSF/1 version 1.2 in a few different configurations. Also included are measurements with TCP and UDP with Digital's patented full-duplex mode for FDDI, which can

potentially increase throughput and reduce latency in FDDI LANs with point-to-point links (which can also be used in switched FDDI LANs). A few implementation ideas currently under study are also presented in the section on Experimental Work.

2 System Characteristics

The project to improve the implementation of Digital's TCP/IP and UDP /IP (the internet protocol suite) networking was targeted on the DEC 3000 AXP Model 500 workstation, running the DEC OSF/1 operating system version 1.2. Since we were interested in achieving the highest performance possible on a commercially available data link, we chose FDDI, and used the DEC FDDIcontroller/TURBOchannel adapter (DEFTA) to communicate between the Alpha AXP workstations. In this section, we describe the features of the workstations, relevant characteristics of FDDI, the internet protocol suite, and the DEC OSF/1 operating system itself, relative to the networking implementation. The architectural features of the Alpha AXP workstations as well as the DEC FDDIcontroller/TURBOchannel adapter are shown in Figure 1.

The Alpha AXP System

The Alpha AXP workstation, DEC 3000 AXP Model 500 was chosen for our research. The system is built around Digital's 21064 64-bit, reduced instruction set computer (RISC) microprocessor.

Digital's 21064 Microprocessor. The DECchip 21064 CPU chip is a RISC microprocessor that is fully pipelined and capable of issuing two instructions per clock cycle.[13,14] The DECchip 21064 microprocessor can execute up to 400 million operations per second. The chip includes

- An 8-kb direct-mapped instruction cache with a 32-byte line size
- An 8-kb direct-mapped data cache with a 32-byte line size
- Two associated translation buffers
- A four-entry (32-byte-per-entry) write buffer
- A pipelined 64-bit integer execution unit with a 32-entry register file
- A pipelined floating-point unit with an additional 32 registers

The DEC 3000 AXP Model 500 Workstation. The DEC 3000 AXP Model 500 workstation is built around the DECchip 21064 microprocessor running at 150 megahertz (MHz).[15] In addition to the on-chip caches, there is an on-board second-level cache of 512 kilobytes (kB). Main memory can

be from 32 MB to 256 MB (1 GB with 16 MB dynamic random-access memories [DRAMs]). The memory bus is 256 bits plus error-correcting code (ECC) wide and has a bandwidth of 114 MB/s. Standard on the system is also a 10-Mb/s Ethernet interface (LANCE). For connection to external peripherals there is an on-board small computer systems interface (SCSI)-2 interface and six TURBOchannel slots with a maximum I/O throughput of 100 MB/s. One of the TURBOchannel slots is occupied by the graphics adapter.

The system uses the second-level cache to help minimize the performance penalty of misses and write throughs in the two relatively smaller primary caches in the DECchip 21064 processor. The second-level cache is a direct-mapped, write-back cache with a block size of 32 bytes, chosen to match the block size of the primary caches. The cache block allocation policy allocates on both read misses and write misses. Hardware keeps the cache coherent on DMAs; DMA reads probe the second-level cache, and DMA writes update the second-level cache, while invalidating the primary data cache. More details of the DEC 3000 AXP Model 500 AXP workstation may be obtained from "The Design of the DEC 3000 AXP Systems, Two High-performance Workstations."[15]

DEC OSF/1 Operating System

DEC OSF/1 operating system version 1.2 for Alpha AXP systems is an implementation of the Open Software Foundation (OSF) OSF/1 version 1.0 and version 1.1 technology. The operating system is a 64-bit kernel architecture based on Carnegie-Mellon University's Mach version 2.5 kernel. Components from 4.3 BSD are included, in addition to UNIX System Laboratories System V interface compatibility.

Digital's version of OSF/1 offers both reliability and high performance. The standard TCP/IP and UDP/IP networking software, interfaces, and protocols remain the same to ensure full multivendor interoperability. The software has been tuned and new enhancements have been added that improve performance. The interfaces between the user application and the internet protocols include both the BSD socket interface and the X/Open Transport Interface.[12] The internet implementation conditionally conforms to RFC 1122 and RFC 1123.[16,17] Some of the networking utilities included are Telnet; file transfer protocol (FTP); the Berkeley "r" utilities (rlogin, rcp, etc.); serial line internet protocol (SLIP) with optional compression; Local Area Transport (LAT); screend, which is a filter for controlling network access to systems when DEC OSF/1 is used as a gateway; and prestoserve, a file system accelerator that uses nonvolatile RAM to improve Network File System (NFS) server response time. The implementation also provides a STREAMS interface, the transport layer interface, and allows for STREAMS (SVID2) and sockets to coexist at the data link layer. There is support for STREAMS drivers to socket protocol stacks and support for BSD drivers to STREAMS protocol stacks via the data link provider interface.

The OSF/1 Network Protocol Implementation

The overall performance of network I/O of a workstation depends on a variety of components: the processor speed, the memory subsystem, the host bus characteristics, the network interface and finally, and probably the most important, software structuring of the network I/O functions. To

understand the ways in which each of these aspects influences performance, it is helpful to understand the structuring of the software for network I/O and the characteristics of the computer system (processor, memory, system bus). We focus here on the structuring of the end system networking code

related to the internet protocol suite in the DEC OSF/1 operating system, following the design of the networking code (4.3 BSD-Reno) in the Berkeley UNIX distribution.[8,9,12]

A user process typically interfaces to the network through the socket layer. The protocol modules for UDP, TCP (transport layers) and IP (network layer) are below the socket layer in the kernel of the operating system. Data is passed between user processes and the protocol modules through socket buffers. On message transmission, the data is typically moved by the host processor from user space to kernel memory for the protocol layers to packetize and deliver to the data link device driver for transmission. The boundary crossing from user to kernel memory space is usually needed in a general-purpose operating system for protection purposes. Figure 2 shows where the incremental overhead for packet processing, based on packet size, occurs in a typical BSD 4.3 distribution.

The kernel memory is organized as buffers of various types. These are called mbufs. They are the primary means for carrying data (and protocol headers) through the protocol layers. The protocol modules organize the data into a packet, compute its checksum, and pass the packet (which is a set of mbufs chained together by pointers) to the data link driver for transmission. From these kernel mbufs, the data has to be moved to the buffers on the adapter across the system bus. Once the adapter has a copy of the header and data, it may return an indication of transmit completion to the host. This allows the device driver to release the kernel mbufs to be reused by the higher layers for transmitting or for receiving packets (if buffers are shared between transmit and receive).

While receiving packets, the adapter moves the received data into the host's kernel mbufs using DMA. The adapter then interrupts the host processor, indicating the reception of the packet. The data link driver then executes a filter function to enable posting the packet to the appropriate protocol processing queue. The data remains in the same kernel mbufs during protocol processing. Buffer pointers are manipulated to pass references to the data between the elements processing each of the protocol layers. Finally, on identifying the user process of the received message, the data is moved from the kernel mbufs to the user's address space.

Another important incremental operation performed in the host is that of computing the checksum of the data on receive or transmit. Every byte of the packet data has to be examined by the processor for errors, adding overhead in both CPU processing and memory bandwidth. One desirable characteristic of doing the checksum after the data is in memory is that it provides end-to-end protection for the data between the two communicating end systems. Because data movement and checksum operations are frequently performed and exercise components of the system architecture (memory) that are difficult to speed up significantly, we looked at these in detail as

candidates for optimization.

The Internet Protocol Suite: TCP/IP and UDP/IP

The protocols targeted for our efforts were TCP/IP and UDP/IP, part of what is conventionally known as the internet protocol suite.[7,9]

TCP is a reliable, connection-oriented, end-to-end transport protocol that provides flow-controlled data transfer. A TCP connection contains a sequenced stream of data octets exchanged between two peers. TCP achieves reliability through positive acknowledgment and retransmission. It achieves flow control and promotes efficient movement of data through a sliding window scheme. The sliding window scheme allows the transmission of multiple packets while awaiting the receipt of an acknowledgment. The number of bytes that can be transmitted prior to receiving an acknowledgment is constrained by the offered window on the TCP connection. The window indicates how much buffering the receiver has available for the TCP connection (the receiver exercises the flow control). This window size also reflects how much data a sender should be prepared to buffer if retransmission of data is required. The size of the offered window can vary over the life of a connection. As with BSD systems, DEC OSF/1 currently maintains a one-to-one correspondence between window size and buffer size allocated at the socket layer in the end systems for the TCP connection. An erroneous choice of window size, such as one too small, or one leading to nonbalanced sender and receiver buffer sizes, can result in unnecessary blocking and subsequent inefficient use of available bandwidth.

TCP divides a stream of data into segments for transmission. The maximum segment size (MSS) is negotiated at the time of connection establishment. In the case of connections within the local network, TCP negotiates an MSS based on the maximum transmission unit (MTU) size of the underlying media. (For IP over FDDI the MTU is constrained to 4,352 octets based on the recommendation in RFC 1390.[18]) TCP calculates the MSS to offer, by subtracting from this MTU, the number of octets required for the most common IP and TCP header sizes.

The implementation of TCP/IP in DEC OSF/1 follows the 4.3 BSD-Reno implementation of TCP. Included is the use of dynamic round-trip time measurements by TCP, which maintains a timer per connection and uses adaptive time-outs for setting retransmission timers. The implementation includes slow start for reacting to congestive loss and optimizations such as header prediction and delayed acknowledgments important for network performance.[19] DEC OSF/1 version 1.2 also includes recent extensions to TCP for accommodating higher-speed networks.[10] TCP's performance may depend upon the window size used by the two peer entities of the TCP connection. The product of the transfer rate (bandwidth) and the round-trip delay measures the window size that is needed to maximize throughput on a connection.

In the TCP specification RFC 793, the TCP header contains a 16-bit window size field which is the receive window size reported to the sender.[9] Since the field is only 16 bits, the largest window size that is supported is 64K bytes. Enhancing the original specification, RFC 1323 defines a new TCP option, window scale, to allow for larger windows.[10] This option

contains a scale value that is used to increase the window size value found in the TCP header.

The window scale option is often recommended to improve throughput for networks with high bandwidth and/or large delays (networks with large bandwidth-delay products). However, it also can lead to higher throughput on LANs such as an FDDI token ring. Increased throughput was observed with window sizes larger than 64K bytes on an FDDI network.

The TCP window scale extension maps the 16-bit window size field to a 32-bit value. It then uses the TCP window scale option value to bit-shift this value, resulting in a new maximum receive window size value. The extension allows for windows of up to 1 gigabyte (GB). To facilitate backward compatibility with existing implementations, both peers must offer the window scale option to enable window scaling in either direction. Window scale is automatically turned on if the receive socket buffer size is greater than 64K bytes. A user program can set a larger socket buffer size via the setsockopt() system call. Based on the socket buffer size, the kernel implementation can determine the appropriate window scale factor.

Similar to the choice of large window sizes, the use of large TCP segments, i.e., those approaching the size of the negotiated MSS, could give better performance than smaller segments. For a given amount of data, fewer segments are needed (and therefore fewer packets). Hence the total cost of protocol processing overhead at the end system is less than with smaller segments.

The internet protocol suite also supports the user datagram protocol or UDP. UDP performance is important because it is the underlying protocol for network services such as the NFS. UDP is a connection-less, message-oriented transport layer protocol that does not provide reliable delivery or flow control. The receive socket buffer size for UDP limits the amount of data that may be received and buffered before it is copied to the user's address space. Since there is no flow control, the UDP receiver may have to discard the packet if it receives a large burst of messages and there is no socket buffer space.

If the receiver is fast enough to allow the user application to consume the data, the loss rate is very low. However, most BSD-derived systems today experience heavy packet loss for UDP even when the receiving processor is the same speed as the transmitter. Furthermore, since UDP has no flow control, there is no mechanism to assure that all transmitted data will be received when the transmitter is faster than the receiver. We describe our implementation of UDP to avoid this behavior, so that packet loss is minimized.

Data Link Characteristics: FDDI

FDDI is a 100 Mb/s LAN standard that is being deployed commercially. It uses a timed-token access method and allows up to 500 stations to be connected with a total fiber length of 200 kilometers. It allows for both synchronous and asynchronous traffic simultaneously and provides a bound for the access time to the channel for both these classes of traffic.

The timed-token access method ensures that all stations on the ring agree to a target token rotation time (TTRT) and limit their transmissions to this target.[20] With asynchronous mode (the most widely used mode in the industry at present), a node can transmit only if the actual token rotation time (TRT) is less than the target.

The basic algorithm is that each station on the ring measures the time since it last received the token. The time interval between two successive receptions of the token is called the TRT. On a token arrival, if a station wants to transmit, it computes a token holding time (THT) as: THT = TTRT - TRT. The TTRT is agreed to by all the stations on the ring at the last time that the ring was initialized (typically happens when stations enter or leave the ring) and is the minimum of the requested values by the stations on the ring. If THT is positive, the station can transmit for this interval. At the end of transmission, the station releases the token. If a station does not use the entire THT allowed, other stations on the ring can use the remaining time by using the same algorithm.

A number of papers relating to FDDI have appeared in the literature, and the reader is encouraged to refer to "Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT," for more details.[21]

Network Adapter Characteristics

The DEC FDDIcontroller/TURBOchannel adapter, DEFTA, is designed to be a high-performance adapter capable of meeting the full FDDI bandwidth. It provides DMA capability both in the receive and transmit directions. It performs scatter-gather on transmit. The adapter has 1 MB of packet buffering. By default, half the memory is used for receive buffering; one fourth of the memory is allocated for transmit buffering; and the remaining memory is allocated for miscellaneous functions, including buffering for FDDI's station management (SMT). The memory itself is not partitioned, and the adapter uses only as much memory as neccessary for the packets. It avoids internal fragmentation and does not waste any memory.

The receive and transmit DMA operations are handled by state machines, and no processor is involved in data movement. The DMA engine is based on the model reported by Wenzel.[22] The main concept of this model is that of circular queues addressed by producer and consumer indices. These indices are used by the driver and the adapter for synchronization between themselves; they indicate to each other the availability of buffers. For example, for receiving packets into the kernel memory, the device driver produces empty buffers. By writing the producer index, it indicates to the adapter the address of the last buffer produced and placed in the circular queue for receiving. The adapter consumes the empty buffer for receiving an incoming packet and updates the consumer index to indicate to the driver the last buffer that it has consumed in the circular queue. The adapter is capable of full-duplex FDDI operation. Finally, FDDI's SMT processing is performed by a processor on board the adapter, with the adapter's receive and transmit state machines maintaining separate queues for SMT requests and responses.

To obtain high performance, communication adapters also try to minimize the amount of overhead involved in transferring the data. To improve performance, the DEFTA FDDI port interface (interface between the hardware and the operating system's device driver) makes efficient use of host memory data structures, minimizes overhead I/O related to the port interface, and minimizes interrupts to the host system.

The Port Architecture contains several unique features that optimize adapter/host system performance. These features include the elimination of much of the control and status information transferred between the host and adapter; the organization of data in host memory in such a way as to provide efficient access by the adapter and the host; and the use of an interrupt mechanism, which eliminates unnecessary interrupts to the host.

The design also optimizes performance through careful organization of data in host memory. Other than the data buffers, the only areas of host memory that are shared by the host and the adapter are the queues of buffer

descriptors and the area in which the adapter writes the consumer indices. The adapter only reads the buffer descriptors; it never writes to this area of host memory. Thus the impact on host performance of the adapter writing to an area in memory, which may be in cache memory, is eliminated. On the other hand, the area in host memory where the adapter writes its consumer

indices is only written by the adapter and only read by the host. Both the receive data consumer index and transmit data consumer index are written to the same longword in host memory, thus possibly eliminating an extra read by the host of information that is not in cache memory. Furthermore, the producer and consumer indices are maintained in different sections of memory (different cache lines) to avoid thrashing in the cache when the host and the adapter access these indices.

The device driver is also designed to achieve high performance. It avoids several of the problems associated with overload behavior observed in the past.[23] We describe some of these enhancements in the next section.

3 Performance Enhancements and Measurements Results

We describe in this section the various performance enhancements included in the DEC OSF/1 operating system version 1.2 for Alpha AXP systems. In particular, we describe the optimizations for data movement and checksum validation, the implementation details to provide good overload behavior within the device driver, the TCP enhancements for high bandwidth-delay product networks, and the UDP implementation enhancements.

We also present measurement results showing the effectiveness of the enhancements. In most cases the measurement environment consisted of two Alpha AXP workstations (DEC 3000 AXP Model 500) on a private FDDI token ring, with a DEC FDDI concentrator. The tests run were similar to the well-known ttcp test suite, with the primary change being the use of the slightly more efficient send and receive system calls instead of read and write system calls. We call this tool inett within Digital. The throughputs obtained were at the user application level, measured by sending at least 10,000 user messages of different sizes. With UDP, these are sent as distinct messages. With TCP, algorithms used by TCP may concatenate multiple messages into a single packet. Time was measured using the system clock with system calls for resource usage. We also monitored CPU utilization with these system calls, and made approximate (often only for relative comparison) conclusions on the usage of resources with a particular implementation alternative.

Optimizations for bcopy() and in_checksum() Routines

In TCP/UDP/IP protocol implementations, every byte of data generally must pass through the bcopy() and in_checksum() routines, when there is no assistance provided in the network interfaces. There are some exceptions: the NFS implementations on DEC OSF/1 avoid the bcopy() on transmit by passing a pointer to the buffer cache entry directly to the network device driver, and UDP may be configured not to compute a checksum on the data. Digital's implementations turn on the UDP checksum by default. Even with the above exceptions, it is important that the bcopy() and in_checksum()

routines operate as efficiently as possible.

To write efficient Alpha AXP code for these routines, we used the following guidelines:

- Operate on data in the largest units possible
- Try to maintain concurrent operation of as many independent processor units (CPU, memory reads, write buffers) as possible
- Keep to a minimum the number of scoreboarding delays that arise because the data is not yet available from the memory subsystem
- Wherever possible, try to make use of the Alpha AXP chip's capability for dual issue of instructions

For network I/O, the bcopy() routine is called to transfer data between kernel mbuf data structures and user-supplied buffers to read()/write() /send()/recv() calls.

The bcopy() routine was written in assembler. This routine always attempts to transfer data in the largest units possible consistent with the alignment of the supplied buffers. For the optimal case, this would be one quadword (64 bits) at a time. The routine uses a simple load/store /decrement count loop that iterates across the data buffer as

Several attempts were made to improve the performance of this simple loop. One design involved unrolling the loop further to perform 64 bytes of copying at a time, while reading ahead on the second cache line. Another involved operating on four cache lines at once, based on concerns that a second quadword read of a cache line may incur the same number of clock delays as the first cache miss, if the second read is performed too soon after the first read. However, neither of these approaches produced a copy routine that was faster than the simple loop described above.

The TCP/UDP/IP suite defines a 16-bit one's complement checksum (in_ checksum()), which can be performed by adding up each 16-bit element and adding in any carries. Messages must (optional for UDP) have the checksum validated on transmission and reception.

As with bcopy(), performance can be improved by operating on the largest units possible (i.e., quadwords). The Alpha AXP architecture does not include a carry bit, so we have to check if a carry has occurred. Because of the nature of the one's complement addition algorithm, it is not necessary to add the carry in at each stage; we just accumulate the carries and add them all in at the end. By operating on two cache lines at a time, we may start the next computation while the carry computation is under way, accumulate all the carries together, then add them all into the result (with another check for carry) at the end of processing the two cache

lines. This results in four cycles per quadword with the addition of some end-of-loop computation to process the accumulated carries. Interleaving the checksum computation across two cache lines also allows for some dual-issue effects that allow us to absorb the extra end-of-loop computation.

DEFTA Device Driver Enhancements

Preliminary measurements performed with the DEC FDDIcontroller/TURBOchannel adapter (DEFTA) and the OSF/1 device driver combination on DEC 3000 AXP Model 500 workstations indicated that we were able to receive the full FDDI bandwidth and deliver these packets in memory to the data link user. Although we show in this paper that the DEC OSF/1 for Alpha AXP system is able to also deliver the data to the user application, we ensure that the solutions provided by the driver are general enough to perform well even on a significantly slower machine. When executing on such a slow system, resources at the higher protocol layers (buffering, processing) may be inadequate to receive packets arriving at the maximum FDDI bandwidth, and the device driver has to deal with the overload. One of the primary contributions of the DEFTA device driver is that it avoids receive livelocks under very heavy receive load.

First, the queues associated with the different protocols are increased to a much larger value (512) instead of the typical size of 50 entries. This allows us to ride out transient overloads. Second, to manage extended overload periods, the driver uses the capabilities in the adapter to efficiently manage receive interrupts. The driver ensures that packets are dropped in the adapter when the host is starved of resources to receive subsequent packets. This minimizes wasted work by the host processor. The device driver also tends to trade off memory for computing resources. The driver allocates page-size mbufs (8K bytes) so that we minimize the overhead of memory allocation, particularly for large messages.

For transmitting packets, the driver takes advantage of the DEFTA's ability to gather data from different pieces of memory to be transmitted as a single packet. Up to 255 mbufs in a chain (although typically the chain is small, less than 5) may be transmitted as a packet. In the unusual case that a chain of mbufs is even longer than 255, we copy the last set of mbufs into a single large page-size mbuf, and then hand the packet to the device for transmission. This enables applications to have considerable flexibility, without resulting in extraneous data movement operations to place data in contiguous memory locations.

In addition, the driver implements a policy to achieve transmit fairness. Although the operating system's scheduling provides fairness at a higher level, the policies within the driver allow for progress on transmits even under very heavy receive overload. Although the Alpha AXP systems are capable of receiving the full FDDI bandwidth, the enhanced transmit fairness may still be a benefit under bursty receive loads during which timely transmission is still desirable. In addition, as transmission links become faster, this feature will be valuable.

Wherever possible, all secondary activities-excluding the transmit and

receive paths-have been implemented using threads. Scheduling secondary activity at a lower priority does not impact the latency of transmit and receive paths.

Improvements to the TCP/IP Protocol and Implementation

The initial TCP window size is set to a default or to the modified value set by the application through socket options. TCP in BSD 4.3 performed a rounding of the socket buffer, and hence the offered window size, to some multiple of the maximum segment size (MSS). The implementation in BSD 4.3 performed a rounding down to the nearest multiple of the MSS. The MSS value is adjusted, when it is greater than the page size, to a factor of the page size.

When using a socket buffer size of 16K bytes, the rounding down to a multiple of the MSS on FDDI results in the number of TCP segments outstanding never exceeding three. Depending on the application message size and influenced by one or more of both the silly window syndrome avoidance algorithms and the delayed acknowledgment mechanism, throughput penalties can be incurred.[16,24]

Our choice in this area was to perform a rounding up of the socket buffer, and hence window size. This enabled existing applications to maintain performance regardless of changes to the buffering performed by the underlying protocol. For example, applications coded before the rounding of the buffer was implemented may have specified a buffer size at some power of 2. We believe it also allows better performance when interoperating with other vendors' systems and provides behavior that is more consistent to the user (they get at least as much buffering as they request).

A buffer size of 4K bytes has long been obsolete for TCP connections over FDDI. Digital chose to increase this buffer to 16K bytes for ULTRIX support of FDDI. With a socket buffer of 16K bytes, even when rounding up is applied, the amount of data is limited to 17,248 octets per round-trip time. We found that the throughput over FDDI is limited by the window size. This is due to the effects of scheduling data packet processing and acknowledgments (ACKs), the interactions with window flow control, and FDDI's token access protocol (described below).[23, 25]

With memory costs decreasing considerably, we no longer consider the 16K byte default to be an appropriate trade-off between memory and throughput. Based on measurements for different values of the window size, we feel that the default window size of 32K bytes is reasonable. Increasing the window size from 16K bytes to 32K bytes results in an increase of the peak throughput over FDDI from approximately 40 Mb/s to approximately 75 Mb/s. However, increasing the window size beyond 32K bytes allowed us to increase the throughput even further, which led us to the incorporation of the TCP window scale extension.

Window Scale Extensions for TCP The implementation of TCP in DEC OSF/1 version 1.2 is based on the BSD 4.3 Reno distribution. In addition, we

incorporated the TCP window scale extensions based on the model proposed in RFC 1323.[10] Our work followed the implementation placed in the public domain by Thomas Skibo of the University of Illinois.

The TCP window scale extension maps the 16-bit window size to a 32-bit value. The TCP window scale option occupies 3 bytes and contains the type of option (window scale), the length of the option (3 bytes), and the "shift-count." The window scale value is a power of 2 encoded logarithmically. The shift-count is the number of bits that the receive window value is right-shifted before transmission. For example, a window shift-count of 3 and a window size of 16K would inform the sender that the receive window size was 128K bytes. The shift-count value for window scale is limited to 14. This allows for windows of (2[16] + 2[14]) = 2[30] = 1 GB. To facilitate backward compatibility with existing implementations, both peers must offer the window scale option to enable window scaling in either direction.

The window scale option is sent only at connection initialization time in an <SYN> segment. Therefore the window scale value is fixed when the connection is opened. Since the window scale option is negotiated at initialization time, only a bit-shift to the window is added to the established path processing and has little effect on the overall cost of processing a segment.

Changes made to the OSF/1 TCP implementation for using the window scale option include the addition of the send window shift-count field and receive window shift-count field to the TCP control block. TCP processing was modified: the receive window shift-count value was computed based on the receive socket buffer size, and the window scale option is sent with the receive window shift-count. A modification at connection initialization time allows the received shift-count value to be stored in the send window shift-count, if TCP receives an <SYN> segment containing a window scale option. The receive window shift-count field is assigned to the window scale option that is sent on the <SYN, ACK> <SYN, ACK> segment. When the TCP enters established state for the connection, window scale is turned on if both sides have sent <SYN> segments with window scale. For every incoming segment, the window field in the TCP header is left-shifted by the send window shift-count. For every outgoing segment, the window field in the TCP header is right-shifted by the receive window shift-count.

Measurement Results with TCP with Alpha AXP Workstations We used the inett tool to measure the throughput with TCP on the DEC OSF/1 operating system between two DEC 3000 AXP Model 500 workstations on a private FDDI ring. We observed that as the window size increased from 32K bytes to 150K bytes, the throughput generally increased for message sizes greater than 3,072 bytes. For example, for a user message size of 8,192 bytes, the throughput with a window size of 32K bytes was 72.6 Mb/s and increased to 78.3 Mb/s for a window size of 64K bytes. The TCP throughput rose to 94.5 Mb/s for a window size of 150K bytes. For window sizes beyond 150K bytes, we did not see a substantial, consistent improvement in throughput between the two user applications in this environment.

We believe that window scale is required to achieve higher throughputs-even in a limited FDDI token ring of two stations - based on the interactions that occur between the token holding time, the scheduling of activities in the operating system, and the behavior of TCP. The default value for TTRT is set to 8 milliseconds.[21] The end system is able to transmit packets at essentially the full FDDI bandwidth of 100 Mb/s, thus potentially consuming about 350 microseconds (including CPU and network interface times) to transmit a maximum-sized FDDI TCP segment of 4,312 bytes. During the 8 milliseconds, the source is able to complete the entire protocol processing of about 23 to 24 segments (approximately 100K bytes).

Further overlap of user data and protocol processing of packets can occur while the data link is transmitting and the sink is generating acknowledgments, if there is adequate socket buffer space in the source system. Thus, with the additional window of approximately 20K bytes to 30K bytes, the source system is able to pre-process enough segments and provide them to the adapter. The adapter may begin transmitting when the token is returned to the sender (after it receives a set of acknowledgments), while the source CPU is processing the acknowledgments and packetizing additional user data. With up to 150K bytes of socket buffer (and hence window), there is maximal overlap in processing between the CPU, the adapter, and the FDDI token ring, which results in higher throughput. This also explains why no further increases in the window size resulted in any significant increase in throughput.

Figure 3 shows the throughput with TCP between two DEC 3000 AXP Model 500 workstations on an isolated FDDI token ring for different message sizes for socket buffer sizes of 32K, 64K, and 150K bytes. For 150K bytes of socket buffer, the peak throughput achieved was 94.5 Mb/s. For all message sizes, we believe that the CPU was not fully utilized. Application message sizes that are slightly larger than the maximum transmission unit size traditionally display some small throughput degradation due to additional overhead incurred for segmentation and the subsequent extra packet processing. We do not see this in Figure 3 because the CPU is not saturated (e.g., approximately 60 percent utilized at message sizes of 8K bytes), and therefore the overhead for segmentation does not result in lower throughput.

So too, application message sizes that are larger than the discrete memory buffer sizes provided by the memory allocator should incur small amounts of extra overhead due to the necessity of chaining such buffers. Figure 3 also shows that the throughput degradation in this case is small.

Improvements to the UDP/IP Protocol Implementation and Measurement Results

UDP is a connection-less, message-oriented transport, with no assurances of reliable delivery. It also does not provide flow control. Unlike TCP,

the UDP transmitter does not buffer user data. Therefore user messages are transmitted directly as packets on the FDDI. When user messages are larger than the MTU size of the data link (4,352 bytes), IP fragments the

data into multiple packets. To provide data integrity, UDP uses the one's complement checksum for both data as well as the UDP header.

In our experience, the receive throughput to applications using UDP/IP with BSD-derived systems is quite poor due to many reasons, including the lack of flow control. Looking at the receive path of incoming data for UDP, we see that packets (potentially fragments) of a UDP message generate a high-priority interrupt on the receiver, and the packet is placed on the network layer (IP) queue by the device driver. The priority is reduced, and a new thread is executed that processes the packet at the IP layer. Subsequently, fragments are reassembled and placed in the receiver's socket buffer. There is a finite IP queue and also a finite amount of socket buffer space. If space does not exist in either of these queues, packets are dropped. Provided space exists, the user process is then woken up to copy the data from the kernel to the user's space. If the receiver is fast enough to allow the user application to consume the data, the loss rate is low. However, as a result of the way processing is scheduled in UNIX-like systems, receivers experience substantial loss. CPU and memory cycles are consumed by UDP checksums, which we enable by default for OSF/1. This overhead in addition to the overhead for data movement contributes to the receiver's loss rate.

Table 1 shows the receive throughput and message loss rate with the original UDP implementation of OSF/1 for different message sizes. We modified the way in which processing is performed for UDP in the receiver in DEC OSF/1 version 1.2. We reorder the processing steps for UDP to avoid the detrimental effects of priority-driven scheduling, wasted work, and the resulting excessive packet loss. Not only do we save CPU cycles in processing, we also speed up the user application's ability to consume data, particularly as we go to larger message sizes.
Table 1 gives the receive throughput and message loss rate with DEC OSF/1 version 1.2 incorporating the changes in UDP processing we have implemented.

Table 1: UDP Receive Characteristics with Peer Transmitter Transmitting at Maximum_Rate

	UDP Receive Before Changes		UDP Receive After Changes	
Message Size (bytes)	Throughput (Mb/s)	Message Loss Rate	Throughput (Mb/s)	Message Loss Rate
128	0.086	98.8%	0.64	83.1%
512	0.354	98.5%	15.14	35.15%
1024	0.394	99.16%	23.77	46.86%
4096	9.5	90.26%	96.91	1.08%
8192	NA*	NA*	97.01	0.56%

* NA: Benchmark did not finish because of significant packet loss in that experiment.

UDP throughput was measured between user applications transmitting and receiving different size messages. Figure 4 shows the throughput at the transmitter, which is over 96 Mb/s for all message sizes over 6,200 bytes and achieves 97.56 Mb/s for the message size of 8K bytes used by NFS. During these measurements, the transmitting CPU was still not saturated and the FDDI link was perceived to be the bottleneck. Therefore, to stress the source system further, we used two FDDI adapters in the system to transmit to two different receivers on different rings. Figure 4 also shows the aggregate transmit throughput of a single DEC 3000 AXP Model 500 workstation transmitting over two FDDI rings simultaneously to two different sinks. The source system is capable of transmitting significantly over the FDDI bandwidth of 100 Mb/s. For the typical NFS message size of 8,192 bytes, the aggregate transmit throughput was over 149 Mb/s. The throughput of the two streams for the different message sizes, indicates that, for the most part, their individual throughputs were similar. This showed that the resources in the transmitter were being divided fairly between the two applications.

Measurements of TCP/IP and UDP/IP with FDDI Full-duplex Mode

Earlier we observed that the behavior of TCP in particular depended on the characteristics of the timed-token nature of FDDI. One of the modes of

operation of FDDI that we believe will become popular with the deployment of switches and the use of point-to-point FDDI is that of full-duplex FDDI. Digital's full-duplex FDDI technology, which is being licensed to other vendors, provides the ability to send and receive simultaneously, resulting in significantly higher aggregate bandwidth to the station (200 Mb/s). More important, we see this technology reducing latency for point-to-point connections. There is no token rotating on the ring, and the station does not await receipt of the token to begin transmission. A station has no restrictions based on the token-holding time, and therefore it is not constrained as to when it can transmit on the data link. The

DEC FDDIcontroller/TURBOchannel adapter (DEFTA) provides the capability of full-duplex operation. We interconnected two DEC 3000 AXP Model 500 workstations on a point-to-point link using the DEFTAs and repeated several of the measurements reported above.

One of the characteristics observed was that the maximum throughput with TCP/IP between the two Alpha AXP workstations, even when using the default 32K bytes window size, reached 94.47 Mb/s. Figure 5 shows the behavior of TCP throughput with full-duplex FDDI operation for different window sizes of 32K, 64K, and 150K bytes (when window scale is used). The throughput is relatively insensitive to the variation in the window size. For all these measurements, however, we maintained the value of the maximum socket buffer size to be 150K bytes. When using a smaller value of the maximum socket buffer size (64K bytes), the throughput drops to 76 Mb/s (for a window size of 32K bytes) as shown in Figure 5.

Although we removed one of the causes of limiting the throughput (token-holding times), full-duplex operation still exhibits limitations due to scheduling the ACK and data packet processing and the resulting lack of parallelism in the different components in the overall pipe (the two CPUs of the stations, the adapters, and the data link) with small socket buffers. Increasing the maximum socket buffer allows for the parallelism of the work involved to provide data to the protocol modules on the transmitter.

Observing the UDP/IP throughput between the DEC 3000 AXP Model 500 workstations, we found a slight increase in the transmit throughput over the normal FDDI mode. For example, the UDP transmit throughput for 8K messages was 97.93 Mb/s as compared to 97.56 Mb/s using a single ring in normal FDDI mode. This improvement is due to the absence of small delays for token rotation through the stations as a result of using the full-duplex FDDI mode.

4 Experimental Work

We have continued to work on further enhancing the implementation of TCP and UDP for DEC OSF/1 for Alpha AXP. We describe some of the experimental work in this section.

Experiments to Enhance the Transmit and Receive Paths for TCP/IP

The bcopy() and in_checksum() routine optimizations minimize the incremental overhead for packet processing based on packet sizes. The protocol processing routines (e.g., TCP and IP) also minimize the fixed per-packet processing costs.

All TCP output goes through a single routine, tcp_output(), which often

follows the TCP pseudocode in RFC 793 very closely.[9] A significant portion of its implementation is weighed down by code that is useful only during connection start-up and shutdown, flow control, congestion, retransmissions and persistence, processing out-of-band data, and so on.

Although the actual code that handles these cases is not executed every time, the checks for these special cases are made on every pass through the routine and can be a nontrivial overhead.

Rather than check each case separately, the TCP/IP code was modified to maintain a bit mask. Each bit in the mask is associated with a special condition (e.g., retransmit, congestion, connection shutdown, etc.). The bit is set whenever the corresponding condition occurs (e.g., retransmit time-out) and reset when the condition goes away. If the bit mask is 0, the TCP/IP code executes straightline code with minimal tests or branches, thus optimizing the common case. Otherwise, it simply calls the original routine, tcp_output, to handle the special conditions. Since the conditions occur rarely, setting and resetting the bits incurs less overhead than performing the tests explicitly every time a packet is transmitted. Similar ideas have been suggested by Van Jacobson.[26]

Additional efficiency is achieved by precomputing packet fields that are common across all packets transmitted on a single connection. For example, instead of computing the header checksum every time, it is partially precomputed and incrementally updated with only the fields that differ on a packet-by-packet basis.

Another example is the data link header computation. The original path involved a common routine for all devices, which queues the packet to the appropriate driver, incurs the overhead of multiplexing multiple protocols, looking up address resolution protocol (ARP) tables, determining the data link formats, and then building the header. For TCP, once the connection is established, the data link header rarely changes for the duration of the connection. Hence at connection setup time, the data link header is prebuilt and remembered in the TCP protocol control block. When a packet is transmitted, the data link header is prefixed to the IP header, and the packet is directly queued to the appropriate interface driver. This avoids the overhead associated with the common routine. Network topology changes (e.g., link failures) may require the data link header to be changed. This is handled through retransmission time-outs. Whenever a retransmit time-out occurs, the prebuilt header is discarded and rebuilt the next time a packet has to be sent.

Some parameters are passed from TCP to IP through fields in the mbufs. Combining the layers eliminates the overhead of passing parameters and validating them. Passing parameters is a nontrivial cost, since in the original implementation, some data was passed as fields in the mbuf structure. Because these were formatted in network byte order, building and extracting them incurred overhead. Moreover, the IP layer does not have to perform checks for special cases that are not applicable to the TCP connection. For example, no fragmentation check is needed since the code for TCP has already taken care to build a packet within the allowed size

limits.

In a similar fashion to the transmit path, a common-case fast path code was implemented for the receive side. This mimics the most frequently executed portions of the TCP/IP input routines, and relegates special cases and errors to the original code. Special cases include fragmented packets, presence of IP options, and noncontiguous packet headers. Combining error checking across TCP and IP also eliminates additional overhead. For example, length checks can be used to detect the presence of options that can be passed to the original general case path.

These fast path optimizations were implemented in an experimental version of the OSF/1 operating system. TCP measurements on the experimental version of OSF/1 running on two systems communicating over a private FDDI ring indicate that, when both the input and output fast path segments are enabled on the two systems, throughput is improved for almost all message sizes.

Experiments to Enhance UDP/IP Processing

An enhancement for UDP/IP processing with which we experimented was to combine the data copying and checksum operations. This has been attempted in the past.[27] The primary motivation is to reduce memory bandwidth utilization and perform the checksums while the data is in the processor during the data movement. To allow us to do this, we introduce a new UDP protocol specific socket option that allows users to take advantage of this optimization. When a user application posts a receive buffer after enabling this socket option, we invoke a combined copy and checksum routine on receiving a packet for that user. In the infrequent case when the checksum fails, we restore the user I/O structure and zero the user buffer so that inappropriate data is not left in a user's buffer. Preliminary performance measurements indicate significant reduction in CPU utilization for UDP receives when using this socket option.

Experiments to Eliminate the Data Copy from User to Kernel Space

As observed earlier, data movement operations add significant overhead on the end system. One method to reduce the cost of data movement for a send operation, prototyped on an experimental version of the OSF/1 operating system, is to replace the data copy from user space to the kernel socket buffer by a new virtual memory page remap function. Instead of copying the data from physical pages in the user map to physical pages in the kernel map, the physical pages associated with the user virtual address range in the user map are remapped to kernel virtual addresses. The pages associated with the new kernel virtual addresses are then masqueraded through the network as mbufs. Preliminary results indicate that a virtual memory mapping technique can be used on the OSF/1 operating system to significantly reduce the overhead associated with the transmission of messages.

The underlying design of the remap operation affects application semantics and performance. The semantics of the application are affected by which underlying page remap operation is selected. Performance may also be affected by the implementation of the page map operation and how well certain TCP/IP configuration variables are tuned to match the processor architecture and the network adapter capabilities.

Two types of remap operations were prototyped: page steal and page borrow. The page steal operation, as the name implies, steals the pages from the user virtual address space and gives the pages to the kernel. The user virtual addresses are then mapped to demand-zero pages on the next page reference. In the page steal operation, the user ends up with demand zero pages. On the other hand, in the borrow page operation, the same physical pages are given back to the user. If the user accesses a page that the kernel was still using, the user process either "sleeps," waiting for that page to become available or (depending upon the implementation) receives a copy of the page. For the page borrow operation, the user buffer size must be greater than the socket buffer size, and the user buffer must be referenced in a round-robin fashion to ensure that the application does not sleep or receive copies of the page.

Both the page steal and the page borrow operations change the semantics of the send() system calls, and some knowledge of these new semantics of the send system calls needs to be reflected in the application. The application's buffer allocation and usage is dependent upon how the underlying remap operation is implemented. An important consideration is the impact on the application programming interface. In particular, the extent to which the semantics of the send system calls (e.g., alignment requirements for the user message buffer) need to change to support the remap operations is an area that is currently under study.

The page remap feature has not yet been incorporated in the DEC OSF/1 version 1.2 product. Inclusion of this feature in the product is expected to reduce CPU utilization. While page remapping does reduce the cost of processing a packet, the design issues outlined above impact applications. To achieve performance benefits and application portability across multiple heterogeneous open systems, future work continues in this area. In addition, integrated hardware solutions to reduce the cost of the copy operation are also under investigation.

The performance numbers presented in this paper did not include the improvements described in this section on experimental work. We anticipate that the overall performance would see substantial improvement with the inclusion of these changes.

5 Conclusions

Increases in communication link speeds and the dramatic increases in processor speeds have increased the potential for widespread use of distributed computing. The typical throughput delivered to applications, however, has not increased as dramatically. One of the primary causes has been that network I/O is intensive on memory bandwidth, and the increases in memory bandwidths have only been modest. We described in this paper an effort using the new Alpha AXP workstations and the DEC OSF/1 operating system for communication over FDDI to remove this I/O bottleneck from the end system.

We described the characteristics of the DEC 3000 AXP Model 500 workstation which uses Digital's Alpha AXP 64-bit RISC microprocessor. With the use of wider access to memory and the use of multilevel caches, which are coherent with DMA, the memory subsystem provides the needed bandwidth for applications to achieve substantial throughput while performing network I/O.

We described the implementation of the internet protocol suite, TCP /IP and UDP/IP, on the DEC OSF/1 operating system. One of the primary characteristics of the design is the need for data movement across the kernel-user address space boundary. In addition, both TCP and UDP use checksums for the data. Both these operations introduce increasing overhead with the user message size and comprise a significant part of the total processing cost. We described the optimizations performed to make these operations efficient by taking advantage of the wider cache lines for the systems and the use of 64-bit operations.

We incorporated several optimizations to the implementation of TCP in the DEC OSF/1 operating system. One of the first was to increase the default socket buffer size (and hence the window size) used by TCP from the earlier, more conservative 4K bytes to 32K bytes. With this, the throughput of a TCP connection over FDDI between two Alpha AXP workstations reached 76.6 Mb/s. By increasing the window size even further, we found that the throughput increases essentially to the full FDDI bandwidth. To increase the window size beyond 64K bytes requires the use of recent extensions to TCP using the window scale option. The window scale option, which is set up at the connection initialization time, allows the two end systems to use much larger windows. We showed that, when using a window size of 150K bytes, the peak throughput of the TCP connection increases to 94.5 Mb/s.

We also improved the performance of UDP through implementation optimizations. Typical BSD-derived systems experience substantial loss at the receiver when two peer systems communicate using UDP. Through simple modifications in the processing for UDP and reordering the processing steps, we improved the delivered throughput to the receiving application

substantially. The UDP receive throughput at the application achieved was 97.56 Mb/s for the typical NFS message size of 8K bytes. Even at this throughput, we found that the CPU of the transmitter was not saturated. When a transmitter was allowed to transmit over two different rings (thus

removing the communication link as the bottleneck) to two receivers, a single Alpha AXP workstation (DEC 3000 AXP Model 500) is able to transmit an aggregate throughput of more than 149 Mb/s for a message size of 8K bytes.

We also described throughput measurements with the FDDI full-duplex mode between two Alpha AXP workstations. With full-duplex mode there are no latencies which are associated with token rotation, lost token recovery, or limitations on the amount of data transmitted at a time as imposed by the FDDI timed-token protocol. As a result, with full-duplex mode there are performance improvements. With TCP, we achieve a throughput of 94.5 Mb/s even with the default socket buffer of 32K bytes. This is smaller than the buffer size needed in token passing mode to achieve the same level of throughput. Since the link becomes the bottleneck at this point, there is no substantial increase in throughput achieved with the use of window scaling when FDDI is being used in full-duplex mode. An increase in peak transmit throughput with UDP is also seen when using FDDI in full-duplex mode.

Finally, a few implementation ideas currently under study were presented.

6 Acknowledgments

This project could not have been successful without the help and support of a number of other individuals. Craig Smelser, Steve Jenkins, and Kent Ferson were extremely supportive of this project and ensured that the important ideas were incorporated into the OSF V1.2 product. Tim Hoskins helped tremendously by providing many hours of assistance in reviewing ideas and the code before it went into the product. In addition, we thank the engineers who ported DEC OSF/1 to Alpha AXP in order to provide a stable base for our work. The DEFTA product development group led by Bruce Thompson and Tom Cassa not only provided us with a nice adapter, but also helped by giving us as many prototype adapters as we needed on very short notice. We would like to thank Gary Lorenz in particular for his help with the DEFTA adapters.

7 References

1. D. Clark and D. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1990, ACM Computer Communications Review, vol. 20, no. 4 (September 1990).

2. J. Lumley, "A High-Throughput Network Interface to a RISC Workstation," Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems, Tucson, AZ (February 17-19, 1992).

3. P. Druschel and L. Peterson, "High-performance Cross-domain Data Transfer," Technical Report TR93-5, Department of Computer Science (Tucson, AZ: University of Arizona, March 1993).

4. G. Chesson, "XTP/PE Overview," Proceedings of the 13th Conference on Local Computer Networks, Minneapolis, MN (October 1988).

5. FDDI Media Access Control, American National Standard, ANSI X3.139-1987.

6. FDDI Physical Layer Protocol, American National Standard, ANSI X3.148-1988.

7. J. Postel, "User Datagram Protocol," RFC 768, SRI Network Information Center, Menlo Park, CA (August 1980).

8. J. Postel, "Internet Protocol," RFC 791, SRI Network Information Center, Menlo Park, CA (September 1981).

9. J. Postel, "Transmission Control Protocol," RFC 793, SRI Network Information Center, Menlo Park, CA (September 1981).

10. V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323, Internet Engineering Task Force (February 1991).

11. K. Ramakrishnan, "Performance Considerations in Designing Network Interfaces," IEEE Journal on Selected Areas in Communications, Special Issue on High Speed Computer/Network Interfaces, vol. 11, no. 2 (February 1993).

12. S. Leffler, M. McKusick, M. Karels, and J. Quarterman, The Design and Implementation of the 4.3 BSD UNIX Operating System (Reading, MA: Addison-Wesley Publishing Company, May 1989).

13. R. Sites, ed., Alpha Architecture Reference Manual (Burlington, MA: Digital Press, 1992).

14. D. Dobberpuhl et al., "A 200-MHz 64-bit Dual-issue CMOS Microprocessor," Digital Technical Journal, vol. 4, no. 4 (Special Issue 1992): 35-50.

15. T. Dutton, D. Eiref, H. Kurth, J. Reisert, and R. Stewart, "The Design of the DEC 3000 AXP Systems, Two High-performance Workstations," Digital Technical Journal, vol. 4, no. 4 (Special Issue 1992): 66-81.

16. R. Braden, "Requirements For Internet Hosts- Communication Layers," RFC 1122, Internet Engineering Task Force (October 1989).

17. R. Braden, "Requirements For Internet Hosts-Application and Support,"

RFC 1123, Internet Engineering Task Force (October 1989).

18. D. Katz, "Transmission of IP and ARP over FDDI Networks," RFC 1390, Internet Engineering Task Force (January 1993).

19. V. Jacobson, "Congestion Avoidance and Control," Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1988, ACM Computer Communications Review, vol. 18, no. 4 (August 1988).

20. R. Grow, "A Timed Token Protocol for Local Area Networks," Presented at Electro/82, Token Access Protocols, Paper 17/3, May 1982.

21. R. Jain, "Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT," Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1990, ACM Computer Communications Review, vol. 20, no. 4 (September 1990).

22. M. Wenzel, "CSR Architecture (DMA Architecture)," IEEE P1212 Working Group Part III-A, Draft 1.3, May 15, 1990.

23. K. Ramakrishnan, "Scheduling Issues for Interfacing to High Speed Networks," Proceedings of Globecom '92 IEEE Global Telecommunications Conference, Session 18.04, Orlando, FL (December 6-9, 1992).

24. D. Clark, "Window and Acknowledgment Strategy in TCP," RFC 813, SRI Network Information Center, Menlo Park, CA (July 1982).

25. L. Zhang, S. Shenker, and D. D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1991, ACM Computer Communication Review, vol. 21, no. 4 (September 1991).

26. V. Jacobson, "Efficient Protocol Implementation," ACM SIGCOMM 1990 Tutorial on Protocols for High-Speed Networks, Part B (September 1990).

27. C. Partridge and S. Pink, "A Faster UDP," Swedish Institute of Computer Science Technical Report, August 1991.

8 General References

E. Cooper, O. Menzilcioglu, R. Sansom, and F. Bitz, "Host Interface Design for ATM LANs," Proceedings of the 16th Conference on Local Computer Networks, Minneapolis, MN (October 1991).

B. Davie, "A Host-Network Interface Architecture for ATM," Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1991, ACM Computer Communication Review, vol. 21, no. 4 (September 1991).

H. Kanakia and D. Cheriton, "The VMP Network Adapter Board (NAB): High Performance Network Communication for Multiprocessors," Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1988, ACM Computer Communication Review, vol. 18, no. 4 (August 1988).

M. Nielsen, "TURBOchannel," Proceedings of 36th IEEE Computer Society International Conference, COMPCON 1991, February 1991.

P. Steenkiste, "Analysis of the Nectar Communication Processor,"

Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems, Tucson, AZ (February 17-19, 1992).

C. Traw, S. Brendan, and J. M. Smith, "A High-Performance Host Interface for ATM Networks," Proceedings of the Symposium on Communications Architectures and Protocols, ACM SIGCOMM 1991, ACM Computer Communication Review, vol. 21, no. 4 (September 1991).

TURBOchannel Developer's Kit, Version 2 (Maynard, MA: Digital Equipment Corporation, September 1990).

9 Biographies

Chran-Ham Chang Chran-Ham Chang is a principal software engineer in the UNIX System Engineering Group and a member of the FAST TCP/IP project team. Since joining Digital in 1987, Chran has contributed to the development of various Ethernet and FDDI device drivers on both the ULTRIX and DEC OSF /1 AXP systems. He was also involved in the ULTRIX network performance analysis and tools design. Prior to this, Chran worked as a software specialist in Taiwan for a distributor of Digital's products. He received an M.S. in computer science from the New Jersey Institute of Technology.

Richard Flower Richard Flower works on system performance issues in multiprocessors, networking, distributed systems, workstations, and memory hierarchies. The need for accurate time-stamping events across multiple systems led him to develop the QUIPU performance monitor. The use of this monitor led to performance improvements in networking, drivers, and RPC. Richard earned a B.S.E.E. from Stanford University (with great distinction) and a Ph.D. in computer science from MIT. Prior to joining Digital, he was a professor at the University of Illinois. Richard is a member of Phi Beta Kappa and Tau Beta Pi.

John Forecast A software consultant engineer with the Networks Engineering Advanced Development Group, John Forecast addresses network performance issues associated with the transmission of audio and video data through existing networks. John joined Digital in the United Kingdom in 1974 and moved to the United States to help design DECnet-RSX Phase 2 products, DECnet Phase IV, and DECnet implementations on ULTRIX and System V UNIX. John also worked on file servers for VMS and a prototype public key authentication system. He holds a B.A. from the University of Lancaster and a Ph.D. from the University of Essex.

Heather Gray A principal engineer in the UNIX Software Group (USG), Heather Gray is the technical leader for networking performance on the DEC OSF /1 AXP product family. Heather's current focus is the development of IP multicast on DEC OSF/1 AXP. She has been involved with the development of Digital networking software (TCP/IP, DECnet, and OSI) since 1986. Prior to joining USG, Heather was project leader for the Internet Portal V1.2 product. She came to Digital in 1984, after working on communication and process control systems at Broken Hill Proprietary Co., Ltd. (BHP) in

Australia.

High-performance TCP/IP and UDP/IP Networking in DEC OSF/1 for Alpha AXP

William R. Hawe A senior consulting engineer, Bill Hawe manages the LAN Architecture Group. He is involved in designing architectures for new networking technologies. Bill helped design the FDDI and extended LAN architectures. While in the Corporate Research Group, he worked on the Ethernet design with Xerox and Intel and analyzed the performance of new communications technologies. Before joining Digital in 1980, Bill taught electrical engineering and networking at the University of Massachusetts, South Dartmouth, where he earned a B.S.E.E. and an M.S.E.E. He has published numerous papers and holds several patents.

Ashok P. Nadkarni A principal software engineer in the Windows NT Systems Group, Ashok Nadkarni is working on a port of native Novell NetWare to Alpha AXP systems. Prior to this, he was a member of the NaC Advanced Development Group. He has contributed to projects dealing with IP and OSI protocol implementations, network performance improvement, a prototype of the Digital distributed time service, and mobile networking. Ashok received a B. Tech. (1983) in computer engineering from the Indian Institute of Technology, Bombay, and an M.S. (1985) from Rensselaer Polytechnic Institute. He joined Digital in 1985.

K.K. Ramakrishnan A consulting engineer in the Distributed Systems Architecture and Performance Group, K.K. Ramakrishnan joined Digital in 1983 after completing his Ph.D. in computer science from the University of Maryland. K.K.'s research interests include performance analysis and design of algorithms for computer networks and distributed systems using queuing network models. He has published more than 30 papers on load balancing, congestion control and avoidance, algorithms for FDDI, distributed systems performance, and issues relating to network I/O. K.K. is a member of the IEEE and the ACM.

Uttam N. Shikarpur Uttam Shikarpur joined Digital in 1988 after receiving an M.S. in computer and systems engineering from Rensselaer Polytechnic Institute. Uttam is a senior engineer and a member of the UNIX Systems Group working on network drivers and data link issues. His current project involves writing a token ring driver for the DEC OSF/1 AXP operating system. Prior to this work, he contributed to the common agent project.

Kathleen M. Wilde As a member of the Networks Engineering Architecture Group, Kathleen Wilde focuses on integration of new TCP/IP networking technologies into Digital's products. For the past two years, she has been prototyping high-performance network features on the OSF/1 operating system and coordinating the standards strategy for Digital's IETF participation. Previously, Kathleen was the development manager of the ULTRIX Network Group. Her responsibilities included product development of TCP/IP enhancements, FDDI, SNMP, and diskless workstation support. She holds a B.S. in computer science and mathematics from Union College.