Understanding Gigabit Ethernet Performance on Sun Fire™ Systems

Jian Huang, Performance/Availability Engineering

Sun BluePrints™ OnLine—February 2003

http://www.sun.com/blueprints

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 817-1657-10
Revision A, 1/8/03
Edition: February 2003

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95045 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.sun.com/patents and one or more additional patents or pending patent applications in the U.S. and in other countries.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Sun BluePrints, Sun Fire, Sun Enterprise, UltraSPARC, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the US and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government Rights —Commercial use. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the Far and its supplements.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, Californie 95045 Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à http://www.sun.com/patents et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats-Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque enregistree aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company Ltd.

Sun, Sun Microsystems, the Sun logo, Sun BluePrints, Sun Fire, Sun Enterprise, SUltraSPARC, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Please
Recycle

Understanding Gigabit Ethernet Performance on Sun Fire™ Systems

Network-centric computing exercises significant pressure on the network performance of servers. With the increasing popularity of gigabit Ethernet, especially the availability of lower-cost, copper-based gigabit Ethernet adapters, the question of how Sun's servers perform in this area is one of the most important issues being addressed by Sun's engineering team.

This article presents an overview of the performance of the Sun™ GigaSwift Ethernet MMF adapters hardware on a Sun Fire™ system in terms of TCP/IP networking. Most of the previous effort on TCP/IP network performance focused on bulk transfer traffic, which imposes on servers a continuous flow of packets with sizes equal to the maximum transfer unit (MTU) of the underlying carrier. In the client-server computing environment, however, not all requests from clients nor all replies from the servers are large. Frequently, the traffic contains packets that are smaller than the MTU of the carrier. Hence, this article investigates the performance of both bulk transfer and small packet traffic on a Sun Fire 6800 server.

This article discusses the network performance of Sun servers and examines the root cause of the network behavior of Sun servers by describing some of the implementation details of the Solaris™ operating environment (Solaris OE). Also, a set of tuning parameters that affect TCP/IP network performance is discussed and some tuning recommendations are made.

Many customers are not familiar with the capability of gigabit Ethernet on the Sun Fire servers. The amount of resource to support gigabit networking on a Sun Fire server is also unknown. In addition, the best practice to tune sun servers for gigabit networking needs to be promoted.

The article presents three levels of detail. The highest level discusses the throughput numbers. The mid-level discusses the amount of resources consumed by gigabit cards and the best practice to tune some of the network parameters. The lowest level discusses the TCP protocol and explains why the system behaves in a certain way.

The audience for this article is primarily Sun resellers, Solaris administrators, and Sun service engineers. The throughput numbers and resource consumption information will help some corporate infrastructure operators (CIOs) and corporate infrastructure architects.

This article covers the following topics:

- *Overview*
- *Categorizing TCP Traffic*
- *Gigabit Ethernet Latency on a Sun Fire 6800 Server*
- *Bulk Transfer Traffic Performance*
- *Small Packet TCP Traffic Performance*
- *Summary*

Overview

Sun servers have been used extensively in the net economy and are powering many popular web sites. Requests from HTTP clients, database clients, mail clients, directory-query clients, and other network service clients exert great pressure on Sun servers through the attached network. The responses from the server also go out through network interfaces. This client-server model of computing depends heavily on the networking performance of the client and the server to provide optimal overall performance.

Current popular network interface cards (NICs) include the fast Ethernet (`hme`) and quadfast Ethernet (`qfe`) cards. These interfaces are only capable of sending and receiving at the 100 megabit-per-second (Mbps) range for each link, which exerts little pressure on the PCI bus bandwidth in a system. However, the newer and faster gigabit Ethernet (GBE) interface cards are gaining momentum. The adoption of GBE has been simplified because the category-5 copper cables, which many of the existing local area networks (LANs) are using, can now carry the gigabit traffic.

Since the major revolution brought by gigabit Ethernet is in throughput (measured in Mbps), this article first focuses on studying bulk transfer type of traffic on Sun Fire servers. However, even though bulk transfer traffic is a major consumer of network bandwidth, a traffic type with a different distribution of packet sizes is more commonly seen when client-server applications are run. Studies show that the sizes of the packets on the World Wide Web (WWW) are trimodal [8]. They can be

about 64 bytes, about 540 bytes, or 1518 bytes (including the four-byte checksum). The maximum Ethernet frame size is 1518 bytes. Hence, restricting the network performance study only to the bulk transfer traffic is insufficient, so this article evaluates the performance of both bulk transfer and small packet traffic.

This evaluation was conducted on the Sun Fire 6800 platform using the Sun™ GigaSwift Ethernet MMF adapters hardware. The article discusses how the network throughput delivered by Sun Fire servers varies with the selected socket buffer sizes, how this throughput scales with the number of CPUs, how the packet rate changes with the application of Nagle's control flow algorithm (discussed in"Small Packet Traffic Issues,") how deferred acknowledgment works, and how long it takes to transmit and receive a packet.

The rest of the article is organized as follows: "Categorizing TCP Traffic" describes the characteristics and performance issues of bulk transfer and small packet traffic and "Gigabit Ethernet Latency on a Sun Fire 6800 Server," presents an evaluation of gigabit network latency. "Bulk Transfer Traffic Performance" and "Small Packet TCP Traffic Performance" discuss the performance of bulk transfer and small packet traffic, respectively. "Summary" concludes the article.

Categorizing TCP Traffic

The maximum frame size of the Ethernet is 1518 bytes. Since Sun uses the version 2 format of Ethernet frame, the Ethernet header is always 14 bytes. Excluding the last four bytes of cyclic redundancy check (CRC) code, the maximum payload for each Ethernet frame, or packet, is 1500 bytes. This is called the MTU size of the Ethernet interface. Since the IP and TCP headers require 20 bytes each when the TCP network is used, the actual payload perceived by the user application is 1,460 bytes for each packet. This 1,460-byte payload is called the TCP maximum segment size (MSS) when Ethernet is the underlying carrier[1]. Based on the MSS, TCP network traffic can be categorized into two types:

- Bulk transfer traffic—The payload size of most segments from the sender to the receiver is 1,460 bytes.
- Small packet traffic—The payload size of most segments from the sender to the receiver is below 1,460 bytes.

In the real life, network traffic is a mixture of both, however, this article presents them separately because they are treated differently in the Solaris OE according to the TCP specifications [5].

1. When TCP options, for example, time stamp, are used, the MSS can be reduced to 1448 bytes.

Typically, bulk transfer traffic is seen when the amount of data to move from one computer to another is far larger than 1,460 bytes. Traffic generated by FTP transfers, network-based backup, and downloading web pages with large amount of graphics fall into this category. Small packet traffic, however, is usually generated by the client/server type of applications. These kinds of applications typically see short requests from the client (for example, a database query), and short replies from the server. (for example, a selected row in the database). Although bulk transfer traffic is the bigger consumer of the two for network bandwidth, both types of traffic require a large amount of system resource to process packets. A system must perform well for both types of traffic to satisfy the demands of the end users, but these two types of traffic exert different types of pressure on the system, and hence behave very differently. The following section describes this behavior.

Bulk Transfer Traffic Performance Issues

The performance of bulk transfer traffic is commonly measured by throughput because the goal is to move as much data as possible in as little time as possible. The overall performance, however, depends on many factors:

- Size of the TCP window
  TCP uses the sliding-windows protocol [4] to control the flow of data and to assist reliable delivery. There are two types of windows. One is the *send* window, the other is the *receive* window. The send window, together with the sender's congestion window [4], helps the sender to avoid congestion. The receive window prevents the sender from overwhelming the receiver with packets by enforcing a limit on how much data can be sent without waiting for acknowledgment. In a switch-based LAN environment, congestion is usually not an issue. Hence, the window typically refers to the receive window. The size of the receive window can be thought of as the size of a pipe between two water-delivering sites. Once a window is negotiated, data from the sender can start to fill the pipe while it flows from the sender to the receiver. When the pipe is full, the sender will stop sending for a while until the receiver drains the pipe to make room. Ideally, in the steady state, data flows at a constant rate from the sender to the receiver. This rate is determined by the slower party between the receiver and the sender. Suppose the rate is X bps, and the latency from sender to the receiver is T seconds (includes the time the sender takes to send a packet, the time the packet travels in the pipe, and the time the receiver takes to process the packet), the pipe must hold at least $X * T/8$ bytes at any given time to ensure that the size of the pipe is not a performance bottleneck. To get one gigabit per second (Gbps) on Ethernet (including the headers), the system must deliver 1,000,000,000/8/1518 = 82,345 packets per second. This is equivalent to delivering one full-sized packet every 12 microseconds. If the latency is 100 microseconds, the size of the window needs to be at least 1,000,000,000 * 0.0001/8 = 12,500 bytes.

- Overhead to move data from the sender to the receiver
  The significance of this issue is two fold. First, if the sender and receiver are incapable of handling 82,345 packets-per-second, reducing overhead helps to improve throughput and packet rate. Second, if the sender and receiver are capable of handling 82,345 packets- per-second, reducing overhead makes the CPUs more available to run other tasks instead of handling the network traffic. The TCP parameters that affect the overhead include, but are not limited to—`tcp_maxpsz_multiplier`, `tcp_deferred_acks max`, and `tcp_deferred_ack` interval.
- Quality of the link
  The quality of the link determines the frequency of dropped/erroneous packets, which leads to retransmitted TCP segments and duplicated acknowledgments (DUP-ACKs). Retransmission and DUP-ACKs waste the effective bandwidth. This article, however, does not evaluate the impact of link quality.

Hence, the general approach to have the best available performance of TCP bulk transfer traffic is to set up a TCP receive window that is sufficiently large and to make the data movement as efficient as possible. Some example mechanisms are selecting the appropriate size for socket buffers, minimizing the number of system calls to send the same amount of data, moving as much data as possible in kernel with each action, and minimizing the number of acknowledgment (ACK) packets from the receiver. “Bulk Transfer Traffic Performance” discusses the details of these mechanisms.

Small Packet Traffic Issues

Request-reply types of applications typically generate small packet traffic. Thus, the latency resulting from packet processing is more important than the throughput delivered. This network latency is usually calculated as measurement time over the number of packets transferred. Hence, if the measurement time is one second, latency is the reciprocal of packet rate-per-second. Since packet rate also shows a server’s capability to process packets besides latency, this article uses this metric for the studies on small packet traffic.

Similar issues such as TCP window size and transfer overhead, discussed previously, also affect the performance of small packet traffic. However, small packet traffic faces other challenges too:

- Nagle’s control flow algorithm
  In RFC 896[5], Nagle’s algorithm is proposed to control congestion and reduce the amount of small packets in the Internet traffic. Small packets are not the most efficient way to transfer data, and, hence, the bandwidth hungry Internet backbone should avoid them as much as possible. Nagle’s algorithm says: “The sender should not transmit the next small packet if it has already one unacknowledged small packet outstanding. Instead, the sender should accumulate small packets until the amount of data to be transferred exceeds the

MTU or when the receiver sends the acknowledgment for the outstanding packet." Hence, applications that send small packets continuously from systems that adopt Nagle's algorithm may observe unwanted delay if the receiving systems enable deferred acknowledgment. However, subsequent sections of this article show that applications can disable Nagle's algorithm on a per connection basis.

- Distinguish the end of a transmission and the traffic generated by small packets For example, a transfer of 1470 bytes results in a full-sized packet (MTU packet) and a packet with 10 bytes of payload is a small packet. Hence, some algorithms targeting quick resolution of end-of-transfer issues may potentially work against the performance of small packet traffic. This issue is not well understood.

Later sections of this article discuss the performance issues of bulk transfer and small packet traffic separately. But before the packet rate and throughput numbers are discussed, how long does a packet travel from the sender's application to the receiver's application?

Gigabit Ethernet Latency on a Sun Fire 6800 Server

As discussed in "Bulk Transfer Traffic Performance Issues," the latency determines the minimal size of the TCP window to achieve one Gbps for bulk transfer traffic. This latency also affects the user-perceived response time. This section discusses the latency for processing packets on the Sun Fire 6800 platform. The TCP Request-Response (TCP RR) tests were conducted using MC-Netperf [3], an extended version of the popular Netperf [2] tool.

The metric for the TCP RR test is transaction rate. A transaction includes these processes—client sending a packet to server (similar to a request), server processing the packet, server sending the client a packet of the same size (similar to a reply), and the client processing the packet. Both the client and the server deal with one request at a time. In this case, both the server and the client are domains of a Sun Fire 6800 server with identical hardware configuration. Both domains use the same driver for the network interface card and the same release (02/02) of the Solaris 8 OE. As a result, assume that the packet processing time is the same for both domains. Hence, a transaction includes two symmetric operations. Each operation includes sending a packet in software, sending a packet in hardware, receiving a packet in hardware, and receiving a packet in software. The latency of one such operation is defined as the latency of gigabit networking on Sun Fire servers. For this article the following data points were collected using one card and one TCP connection for a comprehensive evaluation:

- Latency when the system is conducting bulk transfer. The payload size is 1,460 bytes.
- Latency when the system is handling small packets. Payload size varied from four bytes to 1,024 bytes.

Since the highest transaction rate for the preceding data points indicates the minimum latency, the goal should be to tune the system until it delivers the highest transaction rate. As discussed in the previous section, the setting for deferred acknowledgment, that is, the value of `tcp_deferred_acks_max`[2], may impact the transmission overhead, and hence may impact the transaction rate. However, the unit for `tcp_deferred_acks_max` is in TCP MSS[3]. If the tests only have messages no larger than 1,460 bytes, setting this parameter to one disables deferred acknowledgment, while setting this parameter to any value larger than one enables it. It is intuitive to think that Nagle's algorithm should also have an impact on the transaction rate. However, this algorithm does not govern the timing to send the first small packet, and the TCP RR test will only have one small packet outstanding at any time, thus the impact of Nagle's algorithm can be ignored in this case.

Different settings of Nagle's algorithm and deferred acknowledgment were tested. TABLE 1 shows the results.

TABLE 1 TCP RR Transaction Rates (One Card, Single Connection, Four CPUs)

Payload (bytes)	Deferred ACK off	Deferred ACK on
4	6972	6978
40	6990	6885
100	6714	5358
250	5452	5634
536	4973	5157
1024	4196	4302
1460	3104	3244

The transaction rates peak at 6,978 transactions-per-second and the transaction rates for 4-byte, 40-byte, and 100-byte payloads are very close. Disabling deferred acknowledgments lowers transaction rates for most cases. Using the `snoop` utility shows that both parties in the communication lose the opportunity to piggy-back the ACK in this case. As a result, both parties must process one additional ACK packet for each transaction. Enabling deferred acknowledgment is preferred for this type of traffic. The minimum latency for processing any packets is 6,978 transactions-per-second, which is 140 microseconds round-trip and about 70 microseconds one-way.

2. In Solaris 9 OE, `tcp_local_dacks_max` is used for LAN.

3. In Solaris 9 OE, the unit is in packets.

For packets with a payload of 1,460 bytes, the transaction rate stays about 3109 to 3244 per second. This is roughly 150 microseconds one way. Hence, the latency for gigabit Ethernet on the Sun Fire 6800 is between 70 and 150 microseconds.

Bulk Transfer Traffic Performance

This section investigates the performance of bulk transfer traffic using the Sun GigaSwift Ethernet MMF adapters hardware (driver name `ce`). The goal is to achieve maximum throughput and reduce the overhead associated with data transfer.

Experiment Setup

Before the data is discussed, let's look at the hardware and software environment for this study. The system under test (SUT) is a domain of a Sun Fire 6800 midframe server. It is configured as follows:

- CPU Power—Eight 900 MHz UltraSPARC® III+ processors. Eight gigabytes of memory
- Operating System—Solaris 8 OE Release 02/02
- Network Interface—Sun GigaSwift Ethernet MMF adapters hardware using 66 MHz PCI interface

Five client machines equipped as follows drive the workload:

- Client 1—A second domain of the Sun Fire 6800 server with eight 900 MHz UltraSparc III+ processors using Sun GigaSwift Ethernet MMF adapters hardware.
- Client 2—A Sun Enterprise 6500 with twelve 400 MHz UltraSPARC II processors using Sun Gigabit™ Ethernet adapters P2.0 hardware
- Client 3—A Sun Enterprise 4500 server with eight 400 MHz UltraSparc II processors using Sun Gigabit Ethernet adapters P2.0 hardware
- Client 4—A Sun Enterprise 4500 server with eight 400 MHz UltraSparc II processors using Sun Gigabit Ethernet adapters P2.0 hardware
- Client 5—A Sun Enterprise 450 server with four 400 MHz UltraSparc II processors using Sun™ Gigabit Ethernet adapters P2.0 hardware

All of the gigabit Ethernet cards on client machines use the 66 MHz PCI bus interface. Solaris 8 OE Release 02/02 is running on all of the client machines. Clients are connected to the server using a gigabit switch. MC-Netperf v0.6.1 [3] developed

internally is used to generate the workload. MC-Netperf extends the publicly available Netperf [2] to handle synchronous multiconnection measurements using multicast-based synchronization. Two types of experiments were conducted:

- Single-connection test. The second domain of Sun Fire 6800 (Client 1) is used as the client.
- 10-connection test. Each of the five clients drive two connections.

Runs of 10 minutes are carried out to obtain the numbers discussed in the following section.

Getting an Appropriate Window Size

As described in "Bulk Transfer Traffic Performance" in a switch-based LAN the size of the TCP receive window determines the amount of unacknowledged data the pipe can hold at any given time. The bandwidth delay product determines the pipe size. Since the bandwidth is known to be one Gbps, and the minimal latency is calculated to be 70 to 150 microseconds in "Gigabit Ethernet Latency on a Sun Fire 6800 Server," a pipe must hold at least $1000000000 * 0.000150 / 8 = 18{,}750$ bytes in transit to achieve one gigabit per second (or 964 Mbps excluding Ethernet, IP, and TCP headers) for packets with a 1,460-byte payload. However, whether or not this 964 Mbps number can be achieved depends on a lot of other factors, and the issue of the receive window size is definitely one of the first that needs to be addressed.

TCP Receive Window Tuning Parameters

When a TCP connection is established, both parties advertise the maximum receive window. The *current receive window*, which is the actual receive window during the middle of a transmission, is adjusted dynamically based on the receiver's capability. The *current send window*, although initially small to avoid congestion, ramps up to match the current receive window after the slow start process [4] if the system is tuned properly. A few parameters should be considered in the tuning process. These parameters are considered for the transmit side:

- `tcp_xmit_hiwat`—This parameter is the high watermark for transmission flow control, tunable by using the `ndd` command. When the amount of unsent data reaches this level, no more data from the application layer is accepted until the amount drops to below the `tcp_xmit_lowat` (also tunable by using the `ndd` command). The default value for this parameter is 24,576 in Solaris 8 OE.
- Sending socket buffer size—The sending socket buffer is where the application puts data for the kernel to transmit. The size of the socket buffer determines the maximum amount of data the kernel and the application can exchange in each attempt.

For the reception side, these parameters are considered:

- `tcp_recv_hiwat`—The is the high watermark for reception flow control, tunable by using the `ndd` command. When the application starts to lag behind in reading the data, data starts accumulating in the streamhead. When TCP detects this situation, it starts reducing the TCP receive window by the amount of incoming data on each incoming TCP segment. This process continues until the amount of accumulated data drops to below `tcp_xmit_lowat` (also tunable by using the `ndd` command). The default value for this parameter is 24,576 in Solaris 8 OE[4].
- Receiving socket buffer size—Similar to what the sending socket buffer is for on the transmission side. The receiving socket buffer is where the kernel puts data for the application to read.

The parameter `tcp_xmit_hiwat` determines the default size for the sending socket buffer, so does `tcp_recv_hiwat` for the receiving socket buffer. However, applications can overwrite the default by creating socket buffers of different sizes when calling the `socket` library function. Essentially, tuning for the receive window is equivalent to selecting socket buffer size.

At the first glance, it appears that the size of socket buffers should be set to the largest possible value. However, having larger socket buffers means more resources are allocated for each connection. As discussed earlier, a windows of 18,750 bytes may be sufficient to achieve 964 Mbps. Setting socket buffers beyond a certain size will not produce any more benefit. Furthermore, socket buffer sizes determine only the maximum window size. In the middle of TCP data transmission, the current receive window size is the available buffer space in the receiver, and the current send window size is equal to MIN (receive window, send socket buffer). Hence, the size of the send socket buffer should be no smaller than the receive socket buffer to match the sizes of send window and receive window. In the experiments, the sizes of send socket buffer and receive socket buffer are equal.

The window size only applies to each individual connection. For multiple simultaneous connections, the pressure on window size may not be as large as for a single connection. But what exactly is the value needed for the gigabit Ethernet on Sun Fire servers?

Impact of Socket Buffer Size on Single and 10-Connection Throughput

Socket buffers from 16 kilobytes to one megabyte (note that the size of send socket is always matched with that of receive socket) were investigated using the `ce` interface card. Throughput numbers for both one TCP connection and 10 TCP connections were measured. As FIGURE 1 shows, for a 10-connection sending operation, socket buffers of 48 kilobytes to one megabyte have a significant throughput advantage (up

4. In Solaris 9 OE, the default values for `tcp_xmit_hiwat` and `tcp_recv_hiwat` are 49,152.

to 20 percent improvement) over socket buffers of smaller sizes. However, for a 10-connection receiving operation, only the 24-kilobyte socket buffer is an under performer. Although 16 kilobytes is quite small, the deficiency in individual connections is more than covered by the existence of multiple connections. For the single connection situation, it appears that 48-kilobyte or larger buffers are needed, but 128-kilobyte or larger buffers do not seem to bring additional benefit. In summary, socket buffers of 64 kilobytes appear to be the best compromise to accommodate both single-connection and 10-connection traffic. Future experiments will use 64-kilobyte socket buffers.

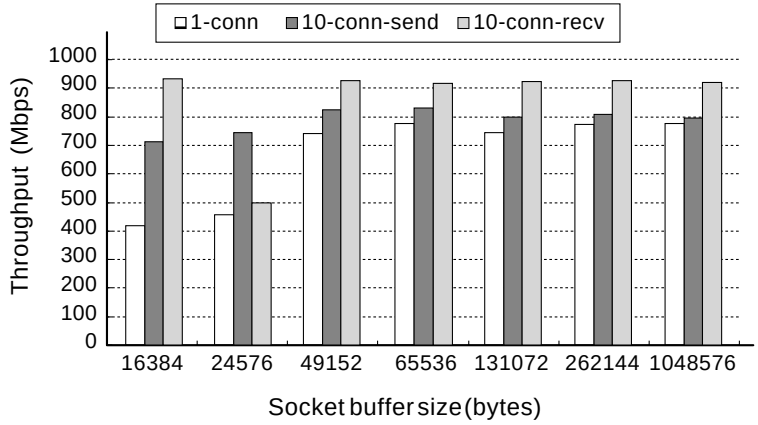

FIGURE 1 Impact of Socket Buffer Size On the Throughput of a `ce` Card

Reducing Transmission Overhead

Assuming the reception side can receive as fast as the sender can transmit, the sender must minimize the overhead it takes to transmit packets. The associated overhead is mostly in moving data between the socket buffer and the kernel modules, and the number of acknowledgment packets the sender needs to process for smooth pumping of data.

Reducing Overhead to Move Data

Since the Solaris OE must copy the data from the application area to the kernel area for transmission, there is an overhead related to the copy operation. The `ndd`-tunable parameter `tcp_maxpsz_multiplier` helps to control the amount of data each copy operation can move (FIGURE 2). Since the TCP module in the kernel needs to process all the pending data before it passes the data down to IP module, this amount should not be too large to prevent the packets from arriving at the hardware

in a continuous flow. The default value for this parameter is two in the Solaris 8 OE and the unit is in TCP MSS. But what exactly is the best value for this parameter to support bulk transfer traffic?

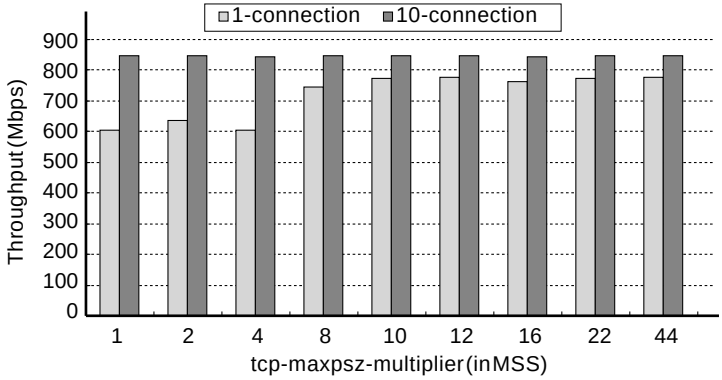

FIGURE 2 Effect of `tcp.maxpsz` Multiplier on Sending Side Performance

To answer this question, 1, 2, 4, 8, 10, 12, 16, 22, and 44 were tried for this parameter using 64-kilobyte socket buffers. Although the maximum allowed value for this parameter is 100, the tests stopped at 44 due to the fact that 64 kilobytes (the socket buffer) divided by 1,460 (the MSS) yields roughly 44.88. As shown in FIGURE 2, this parameter has almost no effect when there are ten connections. For the single-connection case, values of eight or larger deliver about 25 percent higher performance than values of 1, 2, and 4. The best throughput is reached when this parameter is set to 10, so this value is recommended.

Potential Impact of ACK Packets

Another important factor that affects the transmission overhead is the number of acknowledgment (ACK) packets the sender receives for every data packet sent. Each ACK packet is no different than a regular data packet until it reaches the TCP module. Hence, a system must invest a considerable amount of resource to process the ACK packets. The larger the amount of ACK packets, the higher the overhead per data packet.

The ratio of data packets over ACK packets is used to measure this overhead. In Solaris 8 OE, the parameter `tcp_deferred_acks_max` controls the initial maximal amount of data the receiver (in the same local subnet as the sender) can hold before it must emit an ACK packet. Although the unit of this parameter is in TCP MSS, it is equivalent to the number of packets in the bulk transfer traffic. Hence, setting `tcp_deferred_acks_max` (use `ndd`) to eight says the receiver can send one ACK packet for every eight data packets it receives, provided that the timer set by

`tcp_deferred_acks_interval`[5] does not time-out. The effect of `tcp_deferred_acks_max` also depends on the link quality and the status of the network. If ACK packets get lost for some reason, the sender will eventually retransmit data. When the receiver sees this, it will adapt itself to send ACK packets more frequently by reducing the amount of data it can hold without ACKing by one MSS. This process will be triggered for every retransmitted segment. However, in the worst case, the receiver will send an ACK packet for every other data packet it receives, as suggested by RFC 1122.

FIGURE 3 shows how the data packet rate changes when the number of ACK packets gets higher in the first 38 seconds of a connection with poor link quality. This experiment uses the default value of eight (MSS) for the parameter `tcp_deferred_acks_max` and 100 milliseconds for the parameter `tcp_deferred_acks_interval`. The packet rate is the highest when the connection first starts (due to the resolution of one-second the behavior of slow-start phase cannot be observed). Even though eight data packets are desirable for each ACK packet, this figure started with a ratio of four. This ratio is also due to the low resolution of this graph. The packet rate reported is the average packet rate during the past second. Using the `snoop` utility, four retransmitted segments can be observed, which forces the receiver's Solaris 8 OE to adjust four times the amount of data it can hold without ACKing. Two more jumps of the ACK packet rate can be observed in the 6th and the 7th seconds. The ratio of data packets to ACK packets remains about 2.0 for the rest of the connection.

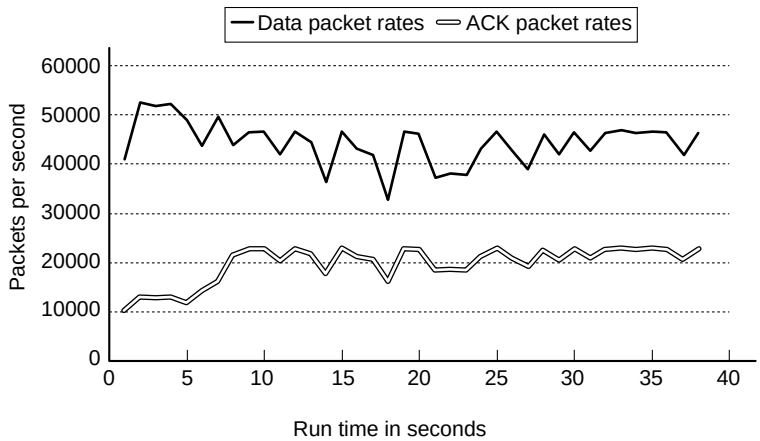

FIGURE 3 Impact of ACK Packets on Data Packet Rate

5. In the Solaris 9 OE, `tcp_local_dack_interval` is used for LAN.

CPU Utilization

Data transfer is only a part of any running user application. To have an application running fast, the CPU time spent in data transfer must be minimized. Hence, knowing how much CPU time is dedicated to data transfer helps to plan the capacity requirement.

The CPU utilization for ce cards when the number of CPUs goes from one to eight was evaluated (FIGURE 4 through FIGURE 7). The utilization numbers shown in these figures are reported as the percentage of time that all available CPUs are engaged. A single number can have different meanings when the underlying number of CPUs is different. For instance, 50 percent utilization for a system with four CPUs means two CPUs are busy on average. Fifty percent utilization for a system with eight CPUs means four CPUs are busy on average.

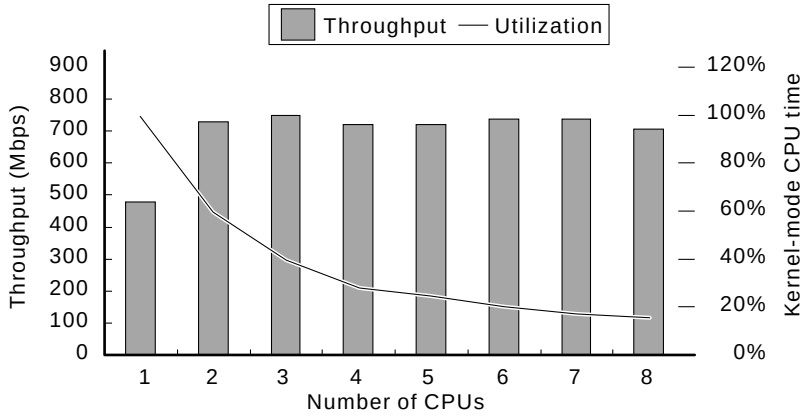

FIGURE 4 Throughput and Amount of Kernel Mode CPU Time Required To Support One TCP Sending Operation By One ce Card

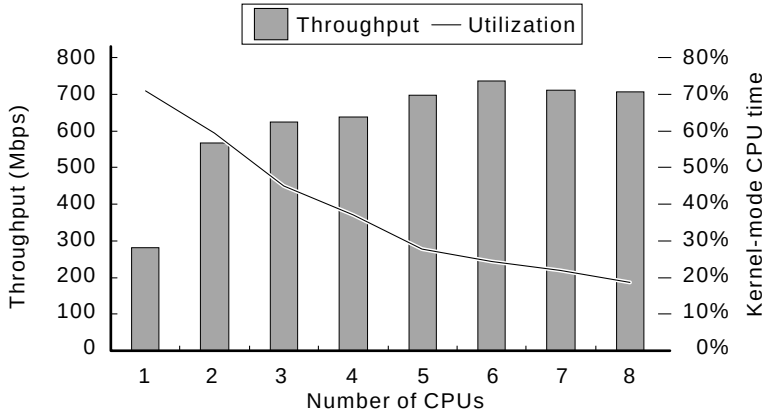

FIGURE 5 Throughput and Amount of Kernel Mode CPU Time Required To Support One TCP Receiving Operation By One ce Card

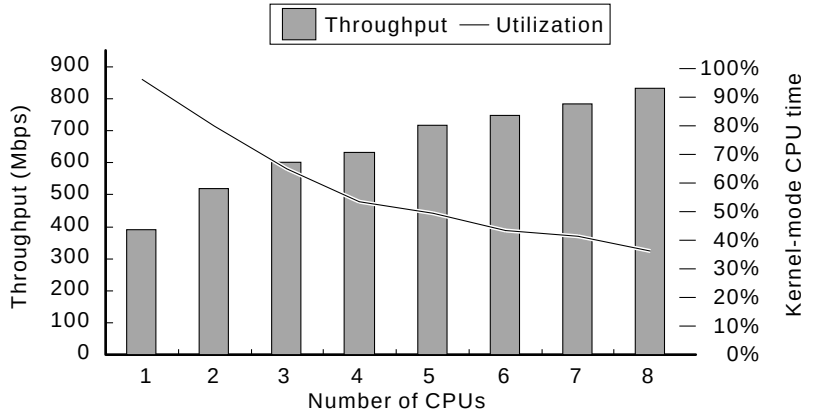

FIGURE 6 Throughput and Amount of Kernel mode CPU Time Required To support 10 Simultaneous Sending Operations By One ce Card

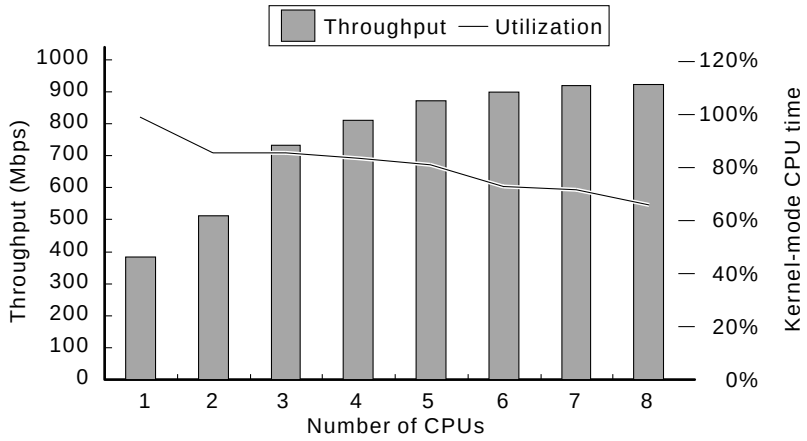

FIGURE 7 Throughput and Amount of Kernel mode CPU Time Required To support 10 Simultaneous Receiving Operations By One ce Card

One-Card, One-Connection CPU Utilization

In the one-connection case[6] (FIGURE 4 and FIGURE 5), two CPUs are sufficient to drive one ce card to its maximum capacity for send-only operations. For reception, three CPUs are necessary to have one ce card deliver 600 Mbps, and five CPUs are needed to deliver 700 Mbps. The best reception performance is achieved with six CPUs (736 Mbps); seven or eight CPUs do not seem to bring additional performance benefit. Even though the overall CPU utilization drops to 15 percent when there are eight CPUs (FIGURE 4), it takes about 8 ∗ 0.15 = 1.2 CPUs to handle network traffic. When there are five CPUs, the number of CPUs to handle network traffic is 5 ∗ 0.26 = 1.30, not far from the 1.2 CPUs needed in the 8-CPU case. In summary, two CPUs are necessary to obtain good performance for one ce card. Diminishing returns are observed for three or more CPUs.

One-Card, Ten-Connection CPU Utilization}

For the 10-connection scenario (FIGURE 6 and FIGURE 7), each additional CPU brings higher sending performance, with eight CPUs achieving 830 Mbps. About 8 ∗ 35% = 2.80 CPUs are dedicated to network traffic. For receiving, ce can reach close to line speed (920 Mbps) with six CPUs, utilizing the power of 6 ∗ 70% = 4.2 CPUs. Diminishing returns are observed when four or more CPUs are added, indicating a recommendation of three CPUs for one ce card.

6. The `/etc/system` parameter `ce_taskq_disable` is set to one to disable the task queues.

Note that the preceding numbers are measured using ce driver version 1.115. The Sun engineering team devotes continuous effort to improving both the throughput and utilization. You can expect to observe better than what is reported in your future experiments.

Small Packet TCP Traffic Performance

This section studies the gigabit performance of TCP with small packet network traffic. The ce interface cards are used for the evaluation. The same set of hardware and software configurations that were used for "Bulk Transfer Traffic Performance" is the test bed. Note that a request/response type of traffic is not investigated, but rather a continuous unidirectional flow of small packets.

Effect of Nagle's Algorithm and Deferred Acknowledgment

As discussed in "Small Packet Traffic Issues," Nagle's algorithm plays an important role in the transmission of small packets. Since Nagle's algorithm asks the sender to accumulate packets when there is one unacknowledged small packet outstanding, packets sent from the application may not be put on the wire as soon as they arrive in TCP. In the meantime, systems at the receiving end typically enable deferred acknowledgment in the hope of having optimal throughput in the bulk transfer case. Hence, if an application is trying to send a series of small messages (less than 1,460 bytes), these messages may not be delivered immediately. Instead, these messages may be delivered with visible delays.

TABLE 2 Performance Of Small Packet Traffic When The Sender Turns On Nagle's Algorithm and the Receiver Enables Deferred Acknowledgment

Message (bytes)	On-wire packet payload (bytes)	Packet rate	Throughput (Mbps)
60	60, 1440	13198	79.20
100	100, 1400	15224	91.35
250	250, 1250	26427	181.56
536	536,1072	36548	235.09
1024	1024	42417	347.49
1460	1460	57146	667.48

This section explains what may be happening, but first looks at the raw packet rate the ce interface is capable of delivering when the sender adopts Nagle's algorithm and the receiver activates deferred acknowledgment. Note that the receiver sets tcp_deferred_acks_max to eight in this case. TABLE 2 lists the packet rate and throughput for a ce card when the server only sends packets. The throughput and packet rate goes up as the message size increases. However, the packet rate of a 60-byte payload is only about one-fifth of the packet rate of a 1,460-byte payload. To understand what causes the low packet rate for small packets, the snoop utility in the Solaris 8 OE was used. FIGURE 8 shows what was found in the 100-byte payload case. After the network ramps up, that is, beyond the slow start [4] phase, a cycle like this is seen. First, the server (Machine S) sends the client (Machine C) a packet with 100 bytes of payload. Machine S cannot continue sending without waiting for an ACK packet from machine C because the last packet it sent had less than 1,460 bytes of payload. Machine C, however, is waiting for more packets from S to reduce the number of ACK packets per data packet. In the mean time, machine S accumulates data from the application. Finally, the amount of unsent data in machine S reaches 1500 bytes (15 packets of 100 bytes), that is, above the 1,460-byte MSS, and sends out a packet with 1400 bytes of payload. Note that machine S will not fragment packets to have a packet with a 1,460-byte payload. When machine C gets the packet with 1400-byte payload, it immediately sends machine S an ACK packet, then this cycle restarts.

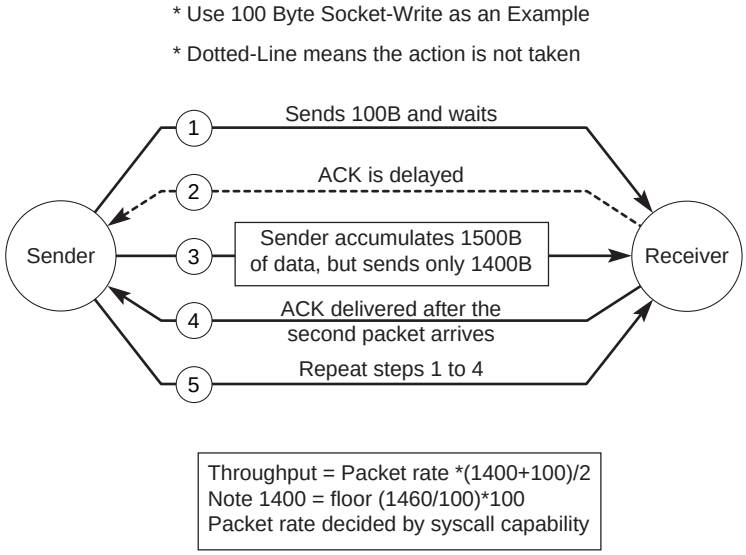

FIGURE 8 Sending Process Of Packets With 100-byte Payload When Nagle's Algorithm is Enabled at the Sender (S) and Deferred Acknowledgment is Enabled at the Receiver (C)

Now one question pops up. Why doesn't C wait until it receives an amount of data no smaller than eight MSS before it sends out an ACK? Obviously, this could have made the interlocking scenario worse, since machine S would have to accumulate another 1500 bytes before it could send out a packet with 1400 bytes of payload. Solaris 8 OE actually takes care of this situation gracefully by enforcing the following rules. In summary, the receiver will send out an ACK packet immediately if both of the following conditions are true:

- A non-MSS segment arrived.
- The amount of unacknowledged data is not a multiple of MSS.

Note that Solaris OE is trying to address the end-of-connection issue here, but it also affects the experiment.

After this scenario is explained, you can see that the relationship of the packet rate and throughput is as follows:

$$Throughput = Packet\_Rate * (floor(1460/message\_size) + 1) * message\_size/2$$

where the *floor* function trims the decimal part of its parameter. The packet rate is mostly determined by the system's capability to execute system calls for moving data from user address space to the kernel address space. Obviously, to obtain better performance for small packets, the system must disable Nagle's algorithm or disable deferred acknowledgment. However, disabling deferred acknowledgment may negatively affect the performance of bulk transfer. Also, the opportunity to piggy-back acknowledgments with data packets from machine C to machine S, if machine C has any, may be lost. Hence, the preferred approach is to disable Nagle's algorithm only in the sender. The two ways to achieve this goal are:

- Set `tcp_naglim_def` to one using the `ndd` command. TCP sends the packet downstream immediately when it receives a message from the application. If the communication involves only Sun servers and workstations, an ACK packet will be delivered after the server transmits two packets.
- In the application, use the `TCP_NODELAY` option in the `socket` function call to create a socket. Only the application can know whether it will be communicating using small packets. Hence, it makes sense to ask the applications that use small packets to disable Nagle's algorithm for the particular sockets that they need. Disabling Nagle's algorithm in a system-wide manner is not preferred.

TABLE 3 shows the new packet rate for the same list of message sizes in TABLE 2 after Nagle's algorithm is disabled. The value of `tcp_maxpsz_multiplier` is set to 10 to produce the numbers in this table. The new packet rates for payloads of 60 bytes to 1,024 bytes increase by up to three times to a level close to the packet rate of the 1,460-byte payload (packets with Ethernet MTU size). Note that even though the packet rates are higher, the actual throughput is lower than the numbers shown in TABLE 2 because each packet now only carries the same amount of payload as the message size. However, no visible pauses will be observed during the transmission. The throughput and packet rate do not change much whether deferred

acknowledgments are enabled or not. Since disabling deferred acknowledgment means higher overhead per data packet and the loss of opportunity to piggyback acknowledgment, disabling this feature is not recommended.

TABLE 3 Performance Of Small Packet Traffic When the Sender Turns Off Nagle's Algorithm

Payload (bytes)	Deferred-ack on packet rate	Throughput (Mbps)	Deferred-ack off packet rate	Throughput (Mbps)
60	37224	25.90	39010	26.36
100	41987	38.15	43736	41.15
250	41297	102.87	42193	99.76
536	43724	188.66	41861	180.15
1024	42576	348.78	41024	336.07
1460	57527	671.92	57554	672.24

Packet Rate Versus Message Size

Traditionally, the packet processing cost is divided into two parts, the cost associated with processing a packet of minimal size, and the cost associated with moving data from the kernel buffer to the user buffer [7]. The former is called *per-packet cost* and the latter is called *per-byte cost*. Under this model, larger packets always take longer to process. This model was developed when the bandwidth of the system backplane was low. However, the current Sun™ Fireplane interconnect can support 9.6 gigabytes per second sustained throughput. This may make the per-byte cost trivial. As a result, the per-packet cost can dominate in processing each packet, which makes the processing time for packets of different sizes very close.

To see the new relationship between packet size and packet rate, the following experiment was conducted:

- Measure packet rates with payload ranging from one byte to 1,460 bytes.
- Tune the system so that one system call from the user application to send a packet corresponds to one packet on wire.
  This is done by disabling the Nagle's algorithm (setting `tcp_naglim_def` to 1).
- Tune the system so that only one packet is moved between kernel components and between kernel and user applications.
  This is done by setting `tcp_maxpsz_multiplier` to 1 on the sending side (server) and setting `tcp_deferred_acks_max` to 1 on the receiving side (client).
- The server only transmits and only one CPU is enabled.

FIGURE 9 shows how the packet rates change when the payload varies from one byte to 1,460 bytes. The numbers shown in this figure are expected to be lower than those shown in TABLE 2 due to the special setting used previously. The packet rates for message sizes of 180 bytes and smaller stay very close for the ce card. The packet rates for message sizes of 250 bytes and beyond are very close also. However, the packet rate for message size of 1,460 bytes (the packets on wire will be full-size Ethernet frames) is only 25 percent lower than those of 180-byte or smaller messages. TABLE 3 shows the percentage of CPU time for user- mode and user-to kernel copy in some of the preceding test cases. Not surprisingly, copy cost is below 10 percent across the board.[7] These findings indicate that the cost associated with copying data in the operating system (which increases more than 350 times from four bytes to 1,460 bytes) is not dominant on Sun Fire 6800 platform. It is the cost associated to process each packet that affects performance most.

TABLE 4 Percentage of CPU Time for User-Mode and Kernel-to-User Copy

Payload (bytes)	Percent user time	Percent kernel to user copy
100	9	1
180	9	2
250	9	3
536	8	5
1024	6	5
1460	6	7

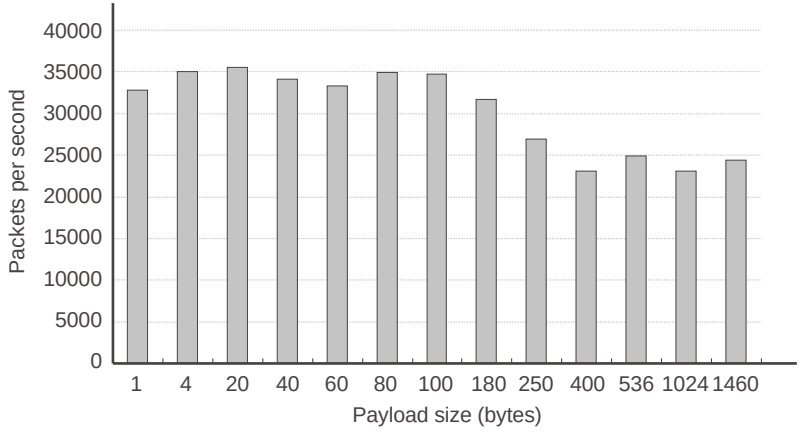

FIGURE 9 Percentage of CPU Time for User-Mode and Kernel-to-User Copy

7. The numbers may be different if throughput varies.

Summary

The increasing popularity of gigabit Ethernet brings new challenges to the network performance. These challenges include both higher bandwidth and higher packet rates. While bulk transfer traffic is more aggressive in bandwidth consumption, small packet traffic is not shy of its demand for the amount of system resources required to process packets. This article discussed the performance for both bulk transfer and small packet traffic separately to reveal some of the issues and the possible solutions related to gigabit Ethernet performance on Sun Fire servers.

Based on the measurements, Sun GigaSwift Ethernet MMF adapters (driver name `ce`) hardware can achieve 738 Mbps for a single TCP connection using an initial TCP window of 48 kilobytes or larger. For 10 TCP connections, one `ce` adapter can achieve 920 Mbps. With per byte cost lowered by the high bandwidth in the backplane, per-packet cost dominates the performance of `ce` cards on Sun Fire servers. To reduce per-packet transmission overhead, a value of 10 for `tcp_maxpsz_multiplier` is recommended in addition to enabling deferred acknowledgment. For send-intensive applications, systems with one `ce` card should have two CPUs. For receive-intensive applications, three CPUs are preferred. Disabling Nagle's algorithm is key to having an acceptable packet rate when the sender is transmitting a series of messages smaller than 1,460 bytes.

About the Author

Jian Huang is currently the lead engineer for the Network Characterization project within Sun's Performance and Availability Engineering (PAE) group. He joined Sun after receiving his Ph.D. degree from the University of Minnesota, based on his thesis research into microprocessor design and compiler-assisted performance enhancements.

Jian has various publications in key academic conferences and renowned international journals within the computer engineering community, including the highly selective IEEE Transactions of Computers. His work at Sun has focused on gigabit network performance, commercial workload characterization, and micro benchmark-based comprehensive network evaluation. He also actively participates in the design of Sun's next-generation microprocessors and network interface cards. Jian has published two papers in the SuperG conference in the past two years.

References

[1] Adrian Cockcroft and Richard Pettit, *Sun Performance And Tuning, Second Edition*, ISBN 0-13-095249-4. Sun Microsystems Press. Also a Prentice Hall Title. Prentice Hall, 1998.

[2] Rick Jones, ”The Public Netperf Home Page,” `http://www.netperf.org`.

[3] Jian Huang, Shih-Hao Hung, Gian-Paolo Musumeci, Miroslav Klivansky, and Keng-Tai Ko, “Sun Fire (TM) Gigabit Ethernet Performance Characterization,”Proceedings of the SuperG Conference, Sun Microsystems, Oct., 2001.

[4] Gary R. Wright, and W. Richard Stevens, *TCP/IP Illustrated, Volume 1*. ISBN 0- 201-63354-X. Addison Wesley, December, 1999.

[5] John Nagle, “Congestion Control in IP/TCP Internetworks,” Request For Comments: 896. January 6, 1984. Network Working Group, Ford Aerospace and Communications Corporation.

[6] R. Braden,”Requirements for Internet Hosts Communication Layers,” Internet Engineering Task Force, Request For Comments: 1122. Oct., 1989.

[7] Hsiao-keng Jerry Chu,”Zero-Copy TCP in Solaris,” Proceedings of the 1996 USENIX Annual Technical Conference, pages 253-264.

[8] K. Chandra and A.E. Eckberg, “Traffic Characteristics of On-Line Services,” Procedings of the 2nd IEEE Symposium on Computers and Communications, Alexandria, Egypt, Huly 1997, pages 17-22.

Ordering Sun Documents

The SunDocs[SM] program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`