Optimistic Implementation of Bulk Data Transfer Protocols

John B. Carter
Willy Zwaenepoel

Department of Computer Science
Rice University
Houston, Texas

Abstract

During a bulk data transfer over a high speed network, there is a high probability that the next packet received from the network by the destination host is the next packet in the transfer. An *optimistic* implementation of a bulk data transfer protocol takes advantage of this observation by instructing the network interface on the destination host to deposit the data of the next packet immediately into its anticipated final location. No copying of the data is required in the common case, and overhead is greatly reduced.

Our optimistic implementation of the V kernel bulk data transfer protocols on SUN-3/50 workstations connected by a 10 megabit Ethernet achieves peak *process-to-process* data rates of 8.3 megabits per second for 1-megabyte transfers, and 6.8 megabits per second for 8-kilobyte transfers, compared to 6.1 and 5.0 megabits per second for the pessimistic implementation. When the reception of a bulk data transfer is interrupted by the arrival of unexpected packets at the destination, the *worst-case* performance of the optimistic implementation is only 15 percent less than that of the pessimistic implementation. Measurements and simulation indicate that for a wide range of load conditions the optimistic implementation outperforms the pessimistic implementation.

This research was supported in part by the National Science Foundation under Grant CCR-8716914 and by a National Science Foundation Fellowship.

1 Introduction

In an *optimistic* implementation of a bulk data transfer protocol, the destination host assumes that the next packet to be received from the network is the next packet in the transfer. The destination host instructs its network interface to deposit the data of the next packet received immediately into its anticipated final location. The packet header is deposited in a reserved buffer area, and is later inspected to confirm that the packet is indeed the next packet in the transfer. If so, the protocol state is updated, but the packet data need not be copied, and the code that handles arbitrary packets is bypassed. If the assumption turns out to be wrong, the data is copied to its correct destination. With a more conventional *pessimistic* protocol implementation, the entire packet is first deposited into a packet buffer. The header is then inspected to decide if this is a packet in a bulk data transfer and to determine the address of the data portion of the packet. Finally, the data portion is copied to its final destination. An optimistic implementation takes full advantage of the *scatter-gather* capabilities of network interfaces such as the AMD LANCE [6] and the Intel i82856 [5] present on various models of SUN workstations. These interfaces allow portions of an incoming packet to be delivered to noncontiguous areas of memory.

The bulk data transfer protocol studied in this paper is a *blast* protocol [11]. The data to be transferred is divided into one or more blasts of fixed maximum size. For each blast, the sender transmits the required number of packets, and then waits for an acknowledgement. The receiver sends back an acknowledgement only after it has received the last packet in a blast. There is no per-packet acknowledgement. Figure 1 presents an example with 2 blasts of 4 packets each. If no acknowledgement is received from the destination, the last packet in the blast is retransmitted until either an acknowledgement is received or the destination is deemed to have failed. Selective retransmission is used to deal with missed packets. Flow control measures may be needed to reduce packet loss when going from a fast to a slow machine [2]. The advantages of a blast protocol over more conventional protocols such as stop-and-wait and sliding window derive from the reduced number of acknowledgements, and from the fact that protocol and transmission overhead on the sender and the receiver occur in parallel [11].

The ideas presented in this paper are not specific to blast protocols, and can be applied to any bulk data transfer protocol in which the following two properties hold:

1. Consecutive packets in a bulk data transfer are likely to arrive from the network at the destination machine uninterrupted by other traffic.

2. The final destination of the data in bulk data packets is known prior to the packets' arrival.

The first property assumes that the average network I/O rate into a machine is low, and that bulk data transfer packets are delivered in short bursts. The second property depends on the protocol's interface with user processes. In a request-response protocol, the destination address for the response data is commonly specified at the time of the request, and hence known before the response arrives. This property also holds for the arrival of the request packet, if the server is ready to receive the packet before the request arrives. In stream protocols like TCP, this property holds if the user *read* operation corresponding to the incoming data is already pending when the data arrives.

Our optimistic implementation of the blast protocols in the V kernel on SUN-3/50s connected by a

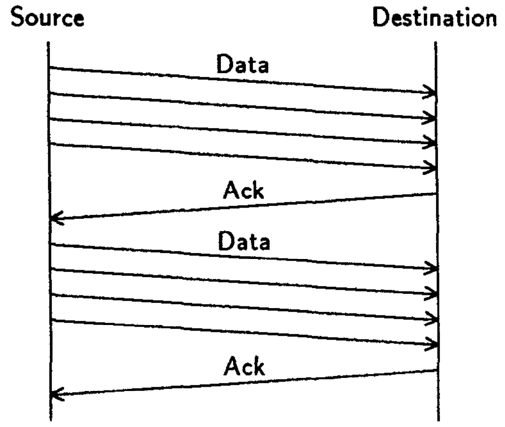

Figure 1 Blast Protocols

10 megabit Ethernet achieves peak *process-to-process* data rates of 8.3 megabits per second for 1-megabyte transfers, and 6.8 megabits per second for 8-kilobyte transfers, compared to 6.1 and 5.0 megabits per second for the original pessimistic implementation. These peak data rates occur when, during a data transfer, consecutive incoming packets are also consecutive packets in the transfer. This property can be disturbed by errors, out-of-sequence packets, and other incoming network traffic. When a blast is interrupted by intervening packets, the worst-case performance of the optimistic implementation is only 15 percent less than the performance of the pessimistic implementation. Measurements on our network indicate that on average 0.8 percent of the blasts directed at workstations and 4.0 percent of blasts directed at the file server are interrupted. Additional experiments show that even under heavier loads at the file server, the percentage of blasts that are interrupted remains low enough that on average the optimistic implementation outperforms the pessimistic implementation.

The outline of the rest of this paper is as follows. Section 2 discusses the implementation of optimistic blast protocols. Section 3 describes an experiment to determine the throughput available through the network interface. In Section 4 we present best-case and worst-case data rates for our optimistic implementation, and compare them to the data rates achieved by a pessimistic implementation. Section 5 describes a series of simulations which predict the performance of an optimistic protocol implementation under various system conditions. In Section 6 we report on the percentage of interrupted blasts observed on our network, and we also discuss the effect of artificially putting a higher load on a shared file server. We discuss related work on locality in network traffic and bulk data transfer protocols in Section 7. In Section 8 we draw conclusions and explore avenues for further work.

2 Implementation

An optimistic implementation of bulk data transfer protocols requires a scatter-gather network interface such as the AMD 7990 LANCE Ethernet interface used on the SUN-3/50 and SUN-3/60. Scatter-gather interfaces allow a single packet to be received in (transmitted from) several noncontiguous locations in memory, thereby avoiding intermediate copies during the reception (transmission) of packets. A pessimistic implementation cannot avoid making a copy at the receiving side. Typically, one or more fields in the packet header indicate where the packet data is to go. Thus, while a pessimistic implementation can receive the packet into noncontiguous locations in memory, it is not possible to determine the final destination of the data without examining the header. This results in a copy of the data, which is usually the largest part of the packet. An optimistic implementation avoids this extra copy except when the bulk data transfer is interrupted or an error occurs.

These interfaces also allow multiple buffers to be queued for reception by means of a *receive buffer descriptor ring*. Each receive buffer descriptor contains an address and a length. The interface deposits incoming packets at the addresses in consecutive buffer descriptors, spreading the packet over multiple receive buffers if the length of the packet exceeds the length indicated in the buffer descriptor. A new packet always starts in a new buffer.

In our implementation, all even-numbered buffer descriptors point to areas in the kernel of length equal to the size of the packet header, and all odd-numbered buffer descriptors point to areas of length equal to the maximum Ethernet packet size minus the size of the header. When no blast reception is in progress, the odd-numbered descriptors point to areas in the kernel such that buffer descriptor $i + 1$ points to the area in memory following that pointed to by buffer descriptor i (See Figure 2). The kernel then operates almost identically to the way it operates without the optimistic implementation, except that it sometimes needs to wait for the data part of a packet to arrive in the second buffer before it has a complete packet. The small packets used for 32-byte message transactions in the V interkernel protocol fit entirely into the header [4].

When the first packet of a blast arrives, the kernel redirects the odd-numbered buffer descriptors to

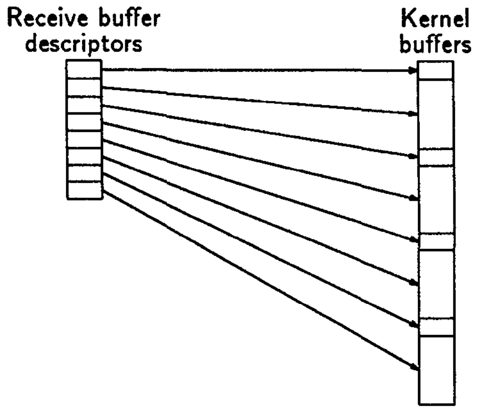

Figure 2 Receive Buffer Descriptors in Default Mode

point to where consecutive blast packets should deposit their data in user memory.[1] The even-numbered buffer descriptors for the headers are not changed (See Figure 3). When handling the interrupt for a header buffer, we check if we received the expected packet, and if so, we simply note its receipt. Interrupts for data buffers require no further action.

When an unexpected packet interrupts a sequence of blast packets, further packets in the blast are deposited at incorrect locations in user memory. We do not attempt to dynamically rearrange the receive buffer descriptors, because this would lead to a complicated race condition between the interface and the kernel. After the last packet of the blast arrives, we send an acknowledgement and copy the blast packets to their correct locations. Thus, a worst-case scenario occurs when an unexpected packet is received immediately after redirection, since this causes all further blast packets to be copied. Because we only redirect buffers for at most 32 kilobytes at a time (the maximum blast size), the effects of an unexpected packet are limited to a single blast and do not spread throughout an entire large transfer.

Packets are also deposited at incorrect locations when one or more packets in the blast are lost. After receiving the final packet in the blast, we copy the data of the erroneously located packets to their intended locations, and request retransmission of the missing packets from the sender, in a manner similar to the selective retransmission strategy used in the pessimistic implementation.

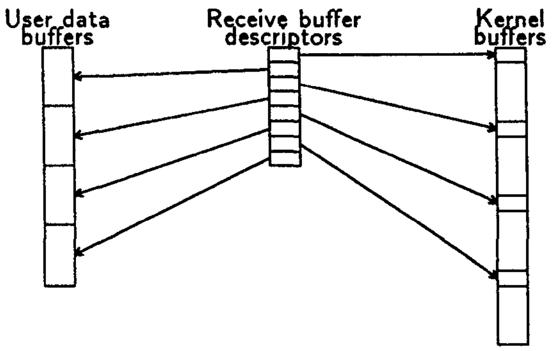

Figure 3 Receive Buffer Descriptors during Blasts

[1]Our current implementation redirects buffers starting at the third packet in the blast, because the hardware is not fast enough to allow us to redirect the second packet in time. The first two packets in a blast are copied to user address space. Also, the kernel must double-map the necessary pages in the receiver's address space into kernel DMA space, because SUN's memory management allows the interface access only to a limited portion of the virtual address space.

The gather capability of a scatter-gather interface allows a packet to be transmitted from noncontiguous locations in memory. We use this capability to construct the packet at the transmitting side without an intermediate copy of the user data into the kernel. A transmit buffer descriptor ring, similar to the receive descriptor ring, is used to queue several packets for transmission. Both our pessimistic and optimistic implementation use the gather capability of the interface, since its use is independent of the optimistic assumption.

3 Hardware Performance

In order to measure the hardware data rate achievable through the LANCE Ethernet interface on the SUN-3/50, we run standalone programs A and B on two machines connected to the network. A sends data to B (possibly in multiple packets), and then B sends the same amount of data to A. The elapsed time of the data transfer is half of the measured time between the first packet leaving A and the last packet arriving at A. The transfers are implemented at the data link layer and at the device level so no protocol or process switching overhead appears in the hardware performance results. In particular, no header other than the Ethernet data link header is added to the data, and no provisions are made for demultiplexing packets, or for retransmission. When a transmission error occurs, the experiment is halted and restarted. Both programs busy-wait on the completion of their current operation, thereby avoiding interrupt handling overhead.

Our measurements show that the hardware can sustain an average raw data rate of 9.4 megabits per second. The actual data rate is somewhat susceptible to other traffic on the network, presumably as a result of the Ethernet's exponential backoff. Packet loss is commonly zero, except when the interface is configured with only a single receive buffer, in which case packet loss is around 40 percent, or with two receive buffers, in which case packet loss averages 0.1 percent. Performance is slightly worse if only a single transmit buffer is used. Otherwise, performance is relatively independent of the number of transmit and receive buffers. Performance is also relatively independent of the number of bytes transferred, as long as the amount transferred is larger than 8 kilobytes.

4 Protocol Performance

Next, we report the *process-to-process* data rates achievable by both the optimistic and the pessimistic implementations of the V kernel bulk data transfer protocol. All measurements are taken by user level processes running on top the V kernel. The data rate reported is the user observed data rate, calculated by dividing the total number of user data bits transferred by the elapsed time. Protocol header bits are not taken into account. Each operation is repeated 1,000 times, and both the average elapsed time and the standard deviation are reported. Time is measured using a software clock accurate to 10 milliseconds.[2]

Table 1 reports measurements of process-to-process data rates observed using our optimistic implementation, when the blast is not interrupted at the destination by other packets and when no errors occur. Under these conditions, the optimistic implementation achieves 88 percent of the hardware data rate for 1-megabyte transfers. The difference between the maximum data rate observed in this experiment and the hardware data rate results from a number of factors, including:

1. The headers (94 bytes long) and the acknowledgements (also 94 bytes long) must be transmitted and consume extra bandwidth. Taking this into account, the total data rate (including both user and header data) for 1-megabyte transfers is 9.1 megabits per second, which is within 5 percent of the hardware data rate.

2. There is a fixed cost for each data transfer, consisting of entry into and exit from the kernel, permission checking, interrupt handling, page map manipulation, and provisions for selective retransmission.

Worst-case behavior occurs when the blast is interrupted immediately after the buffers are redirected. We measure this case by modifying the reception routine so that the check to verify that the correct packet is received always fails. This causes

Size (Kbytes)	Elapsed Time (msec.)		Rate (Mbps)	% of Hardware Data Rate
	Mean	Dev.		
4	6.2	0.2	5.3	56%
8	9.7	0.5	6.8	72%
32	33.0	1.0	8.0	85%
1024	1015	19	8.3	88%

Table 1 Best-Case Optimistic Implementation

[2]We repeat small transfers 20 times to guarantee that each measurement is much larger than the clock period.

the reception routine to always invoke its error recovery routine (including making a copy of the data in the packet). Table 2 indicates the performance of the optimistic blast protocol implementation under these circumstances. Note that there are two lines in this table corresponding to a 1-megabyte transfer. The first line refers to the extremely unlikely case that every 32-kilobyte blast during the transfer is interrupted in a worst-case manner. The second line refers to the case where only a single one of the 32-kilobyte blasts is interrupted in a worst-case manner and the others are not interrupted.

In Table 3 we provide the data rates achieved by a pessimistic implementation of the same protocol.[5] Comparison of Tables 1 and 3 shows the benefits of the optimistic implementation under favorable circumstances. These benefits derive mainly from avoiding copying the data except for the first two packets in the blast.[6] Permission checking and packet handling are also greatly simplified in the case when the expected packet arrives. Comparison of Tables 2 and 3 shows that the performance loss caused by an erroneous guess is relatively minor, even in the worst case. We only need to copy the data from where we assumed it would land to its correct destination. In a pessimistic implementation, the data must always be copied from the kernel receive buffer area to its destination. The extra overhead results from setting up the appropriate structures for the optimistic implementation and releasing them. Figure 4 provides a graphical comparison of the throughput achieved by the various implementations.

When two 1-megabyte data transfers are taking place simultaneously between two pairs of machines, each transfer receives roughly half the total available network throughput (See also [1]). When 1-megabyte data transfers from two different machines are directed simultaneously at the same destination machine, the transfers achieve an average total throughput of 6.5 megabits per second. 35 percent of these blasts are interrupted by a blast packet from a different blast.

Packet loss is uncommon in our implementation. A large percentage of the packet loss on an Ethernet occurs as a result of receive buffer overflow. Our implementation is configured with 64 receive buffers, enough to avoid noticeable packet loss. We intend to study the packet loss issue further in connection with flow control for transfers from a fast to a slow machine.

Size (Kbytes)	Elapsed Time (msec.)		Rate (Mbps)	% of Hardware Data Rate
	Mean	Dev.		
4	7.8	0.4	4.2	45%
8	14.4	0.6	4.6	49%
32	52.3	0.9	5.0	53%
1024 [3]	1667	22	5.0	53%
1024 [4]	1035	—	8.1	86%

Table 2 Worst-Case Optimistic Implementation

[3]Each 32-kilobyte blast is interrupted.

[4]Only one 32-kilobyte blast is interrupted.

[5]These measurements are better than those reported in [3], because the latter implementation did not use the gather capability of the interface during transmission, while ours does.

[6]A memory-to-memory copy takes 0.37 milliseconds per kilobyte.

Size (Kbytes)	Elapsed Time (msec.)		Rate (Mbps)	% of Hardware Data Rate
	Mean	Dev.		
4	7.8	0.4	4.2	45%
8	13.1	0.5	5.0	53%
32	44.4	1.0	5.9	63%
1024	1376	20	6.1	65%

Table 3 Pessimistic Implementation

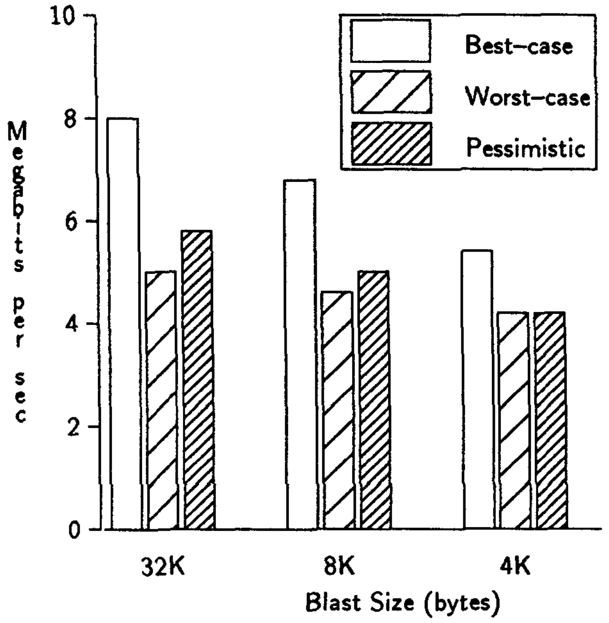

Figure 4 Throughput vs Transfer Size

5 Simulated Performance

We use simulation to study the performance of the optimistic blast protocol implementation over a wide range of system conditions. We assume that the interarrival times for blasts and for non-blast packets are exponentially distributed, and that the non-blast packets are independent of the blast transfers. The experiments consist of a series of terminating (finite horizon) simulations, each with a period of 500 simulated seconds. The experiments are repeated a sufficient number of times to construct a confidence interval on the percentage of blasts interrupted with a 95 percent approximate confidence and a relative precision of 5 percent. The resulting performance predictions match reasonably well with observed performance (See Section 6).

Figures 5 and 6 present the results of a series of experiments designed to study the performance of an optimistic blast protocol implementation on a workstation. We assume that the blasts are transmitted from a single source (i.e., the file server), and thus cannot interrupt one another. The blast arrival rate is chosen so that an average of 8 kilobytes of blast data arrive per second. Figure 5 shows the probability that a blast of a given size is interrupted as a function of the arrival rate of non-blast packets, for blast sizes of 4, 8, and 32 kilobytes. Figure 6 gives the resulting throughput, which is calculated in the following way. For uninterrupted blasts, we take the best-case elapsed time from Table 1. For interrupted blasts, we calculate the elapsed time as the worst-case elapsed time from Table 2 minus the time to copy half of the number of redirected buffers,[7] since on average the blast is interrupted halfway through, after a number of packets have been received without a copy.

Figure 7 presents the results of a series of experiments designed to study the performance of an optimistic blast protocol implementation on a file server. Here, we assume that the blasts are transmitted from multiple sources, so that blasts can interrupt one another. We choose a constant blast size of 8 kilobytes, which is the predominant size of blasts arriving at our file server. Figure 7 shows the percentage of interrupted blasts as a function of the arrival rate of non-blast packets for blast traffic rates of 0.5, 2, 8, and 32 kilobytes per second.

[7]Redirection in the simulation begins with the third packet.

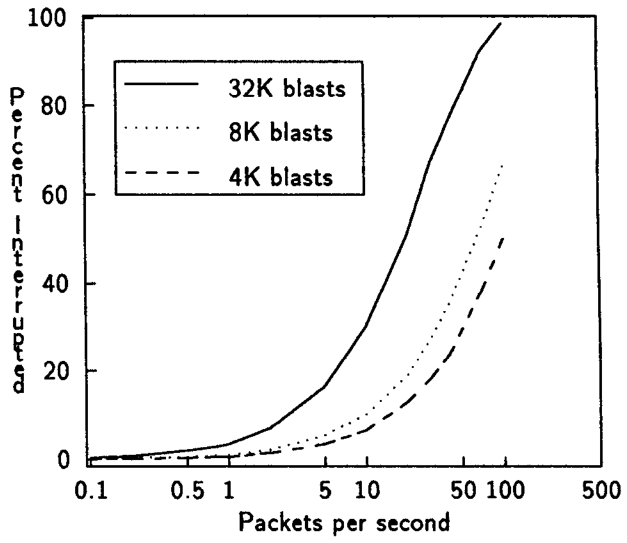

Figure 5 Percentage of Blasts Interrupted at a Workstation

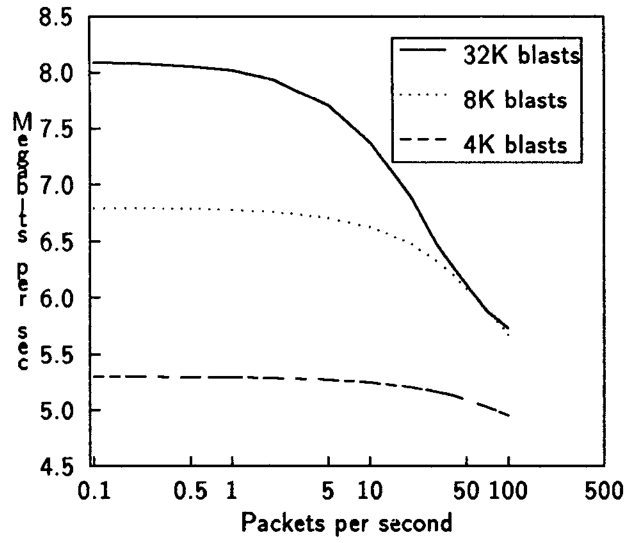

Figure 6 Throughput at a Workstation

6 Performance in an Operating Environment

6.1 Observed Environment

Our Ethernet connects over 60 machines, mostly diskless SUN workstations, 8 file servers, and a few large machines. Virtually all of the machines are used for research, software development, and text processing. Additionally, the network has gateways to the ARPANET, the NSF regional and backbone networks, and the campus backbone network. These gateways are responsible for a large part of the broad-

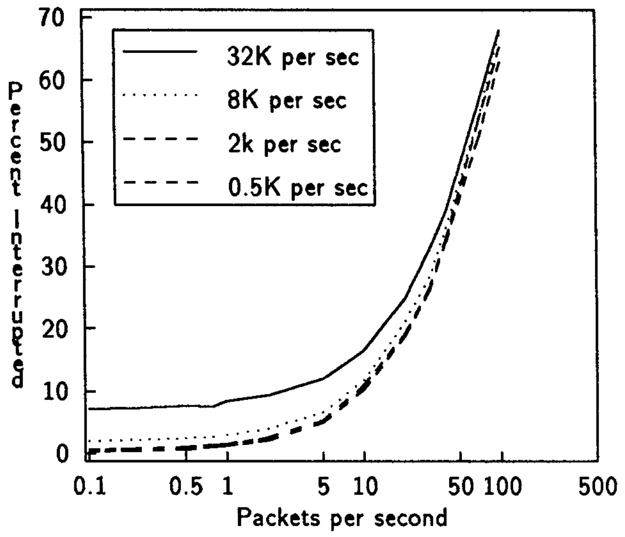

Figure 7 Percentage of Blasts Interrupted at a File Server (8K blasts)

cast and multicast traffic on the network (approximately 0.8 packets per second). The network is heavily loaded. The network load averaged over 1-minute intervals fluctuates between 5 and 30 percent.

Between 8 and 12 diskless SUN workstations run the V-System. They are supported by a V-based SUN-3 file server. Most V network traffic is between workstations and the file server, although there is some workstation-to-workstation traffic.[8] The file server uses an 8-kilobyte page size. The following statistics were gathered for each machine:

- The total number of blasts received.
- The number of interrupted blasts, and the type of packet that caused the blast to be interrupted.
- The average blast size.
- The number of non-blast packets received, including non-blast V, other unicast, and broadcast packets.

6.2 Normal Operation

To assess performance during normal operation, we have collected statistics for different periods of 10,000 seconds each. While the exact numbers differ somewhat from one session to another, the overall shape of the figures remains the same. Table 4 contains the results of a representative monitoring session. The column labeled WS represents measurements taken on workstations, and the column marked FS represents measurements taken on the file server.

[8]For a detailed account of V network traffic see [3].

	WS	FS
Number of Blasts	3237	789
Number of Interrupted Blasts	27	32
Percentage of Blasts Interrupted	0.8%	4.0%
Interruptions by Broadcasts	4	1
Interruptions by V Blasts	0	8
Interruptions by Other V packets	23	23
Blasts per Second	0.3	0.1
Other Packets per Second	1.4	3.7
Average Blast Size (kilobytes)	7.0	6.9

Table 4 Normal Load

The percentage of blasts interrupted matches quite well with the predictions from the simulation. At a workstation, an average of 1.4 unrelated packets arrive per second, and the average incoming blast size is 7.0 kilobytes. The simulation predicts that for 8-kilobyte blasts and an unrelated packet arrival rate of 1.4 packets per second, 1.2 percent of the blasts should get interrupted, vs. 0.8 percent as observed. Similarly, for the observed file server traffic, the simulation predicts that 3.7 percent of the blasts will get interrupted, vs. 4.0 percent as observed.

We conclude that under normal operation, the precentage of interrupted blasts directed at workstations is extremely low. The file server is the target of more unrelated traffic from different sources, and hence the percentage of interrupted blasts is somewhat higher, but it is still low enough that the actual performance approximates the best-case performance.

6.3 High Load Experiment

A high load situation on the file server is created by running a 15-minute sequence of compilations on N diskless SUN-3/50s ($N = 1, ..., 5$). Tables 5 and 6 present the results of these experiments. In Table 5 we indicate the number of incoming blasts, the number of incoming non-blast packets, and the percentage of blasts interrupted, both per workstation and for the file server. Table 6 presents some cumulative statistics for all of the experiments.

We compare these results with those predicted by the simulation. For workstations, the correspondence is still quite good. For example, for the experiment with 5 workstations, the measurements indicate 2.0 percent vs. 2.4 percent predicted by the simulation. The prediction for the file server is less accurate: 11.4 percent measured vs. 16.2 percent predicted. In reality, blasts arriving at the file server are not totally independent, and blasts and non-blast packets are

N	Blasts (per sec.)		Non-Blast Packets (per sec.)		Percentage of Blasts Interrupted	
	WS	FS	WS	FS	WS	FS
1	2.52	0.60	2.21	5.67	2.3%	4.1%
2	2.33	1.03	2.45	7.21	2.1%	6.3%
3	1.98	1.21	2.69	10.13	2.2%	6.8%
4	2.05	1.73	2.98	13.10	2.2%	9.2%
5	2.17	2.14	3.06	16.23	2.0%	11.4%

Table 5 High Load Experiment

	WS	FS
Number of Blasts	28953	6039
Number of Blasts Interrupted	637	477
Percentage of Blasts Interrupted	2.2%	7.9%
Interruptions by Broadcasts	180	24
Interruptions by V Blasts	0	36
Interruptions by Other V packets	457	417

Table 6 High Load Experiment
Cumulative Statistics

somewhat correlated as well. This correlation seems to reduce the percentage of blasts interrupted from what the simulation predicts.

Even for the largest configuration considered, the "average" performance of our optimistic implementation is still significantly better than the pessimistic implementation. For instance, using the figure in Table 5 of 11.4 percent interruptions on the file server, assuming 8-kilobyte transfers, and assuming a worst-case scenario for each interrupted blast, the average data rate becomes 7.6 megabits per second (from Tables 1 and 2), which is significantly better than the data rate of 5.0 megabits per second achieved by the pessimistic implementation (See Table 3). The numbers reported here are conservative estimates for blast interruptions at a file server because we do not use any caching on the workstations. This significantly increases the traffic to the file server, and hence the number of interrupted blasts.

7 Related Work

Locality in network traffic has been noted earlier [3, 8, 10]. However, to the best of our knowledge, no protocol implementation has taken advantage of this phenomenon to the extent described here. In his efforts to improve TCP performance, Jacobson predicts that the next incoming TCP packet on a given connection will be the next packet on that connection [7]. This prediction is less aggressive than ours, which assumes that the next packet that is received at a host is the next in the current bulk data transfer. In the expected case, Jacobson's TCP implementation avoids invoking the general purpose packet reception code, but the data must still be copied to user space.

The blast protocols discussed here were developed as part of the V interkernel protocol [4]. Similar protocols are now part of the VMTP protocol definition [3]. VMTP claims 4.5 megabits per second process-to-process data rates on SUN-3/75s (slightly faster than our SUN-3/50s), with no performance improvements when increasing the segment size beyond 16 kilobytes. The current VTMP implementation uses neither the scatter nor the gather feature of the network interfaces on the SUN.

Sprite uses blast protocols (called implicit acknowledgement and fragmentation) for multi-packet RPCs [9]. Sprite performance measurements, also on SUN-3/75s, indicate a kernel-to-kernel data rate of approximately 6 megabits per second, and a process-to-process data rate of 3.8 megabits per second. We speculate that the difference between the kernel-to-kernel and process-to-process data rates is largely due to an extra copy between kernel and user address space, further emphasizing the need for avoiding this copy as in our optimistic implementation.

8 Conclusions

We have described an optimistic implementation of a bulk data transfer protocol, which predicts that during receipt of a blast the next packet that comes in from the network is the next packet in the blast. The implementation instructs the network interface to deposit the data of the next packet in the location for the data in the next bulk data transfer packet. If the prediction is correct, an optimistic implementation avoids an extra data copy.

We have presented both best-case and worst-case performance, and compared them with the performance of a pessimistic implementation. Both simulation and experience indicate that the actual performance of the optimistic implementation is significantly better than the performance of the pessimistic implementation, even with a relatively high network load on a shared server machine.

More work is required on network interfaces that facilitate optimistic implementation of bulk data transfers. Also, flow control is necessary when transmitting from a fast to a slow machine. We intend to experiment with adapting the blast size as a function of the speed of the receiving machine, while still transmitting at full speed. Hopefully, this will allow uninterrupted packet arrival without overrunning the receiver's buffers. Finally, we want to extend our optimistic blast implementation to allow multicast delivery of bulk data.

References

[1] D.R. Boggs, J.C. Mogul, and C.A. Kent. Measured capacity of an Ethernet: Myths and reality. In *Proceedings of the 1988 Sigcomm Symposium*, pages 222–234, August 1988.

[2] D.R. Cheriton. VMTP: A transport protocol for the next generation of communication systems. In *Proceedings of the 1986 Sigcomm Symposium*, pages 406–415, August 1986.

[3] D.R. Cheriton and C.L. Williamson. Network measurement of the VMTP request-response protocol in the V distributed system. In *Proceedings of the 1987 ACM Sigmetrics Conference*, pages 128–140, October 1987.

[4] D.R. Cheriton and W. Zwaenepoel. The distributed V operating system and its performance for diskless workstations. In *Proceedings of the Ninth ACM Symposium on Operating Systems Principles*, pages 128–140, October 1983.

[5] Intel Corporation. Intel i82856 interface reference guide.

[6] Advanced Micro Devices. Am 7990: Local area network controller for Ethernet (LANCE).

[7] V. Jacobson. Note on TCP/IP mailing list, tcp-ip@SRI-NIC.ARPA, March 1988.

[8] R. Jain and S. Ruthier. Packet trains: Measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communication*, SAC-4(6):986–995, September 1986.

[9] J.K. Ousterhout, A.R. Cherenson, F. Douglis, M.N. Nelson, and B.B. Welch. The Sprite network operating system. *IEEE Computer*, 21(2):23–36, February 1988.

[10] J.F. Shoch and J.A. Hupp. Measured performance of an Ethernet local network. *Communications of the ACM*, 23(12):711–721, December 1980.

[11] W. Zwaenepoel. Protocols for large data transfers over local area networks. In *Proceedings of the 9th Data Communications Symposium*, pages 22-32, September 1985.