An Evaluation of Object-Based Data Transfers on High Performance Networks

Phillip M. Dickens
Department of Computer Science
Illinois Institute of Technology
dickens@iit.edu

William Gropp
Mathematics and Computer Science Division
Argonne National Laboratory
gropp@mcs.anl.gov

Abstract

In this paper, we describe FOBS: a simple user-level communication protocol designed to take advantage of the available bandwidth in a high-bandwidth, high-delay network environment. We compare the performance of FOBS with that of TCP both with and without the so-called Large Window extensions designed to improve the performance of TCP in this type of network environment. It is shown that FOBS can obtain on the order of 90% of the available bandwidth across both short and long high-performance network connections. In the case of the long haul connection, this represents a bandwidth that is 1.8 times higher than that of the optimized TCP algorithm. Also, we demonstrate that the additional traffic placed on the network due to the greedy nature of the algorithm is quite reasonable, representing approximately 3% of the total data transferred.

1. Introduction

A very important area of current research is the development, implementation, and testing of the cutting-edge networking infrastructure of the future (e.g. Abilene [20] and VBNS [21]). An integral component of such research efforts is the development and testing of high-performance distributed applications that, due to the limited bandwidth and best-effort nature of the Internet1 environment, were heretofore infeasible. Examples of such applications include distributed collaboration across the Access Grid, remote visualization of terabyte (and larger) scientific data sets, high-performance computations executing on the Computational Grid, Internet telephony, and multimedia applications. The high-performance networks currently being developed and tested offer the promise of connectivity at speeds of up to 40 Gigabits per second. Clearly such high-performance networks, and the advanced distributed applications that are and will be developed to execute on top of this networking infrastructure, will become a critical component of the national computational infrastructure.

At the most fundamental level, the development of a very high-performance national-scale (and even international-scale) networking infrastructure is about the ability to transfer vast quantities of data, across geographically distributed computational environments, in a very efficient manner. All of the advanced networking applications currently being developed, as well as those envisioned for the future, are based upon this fundamental premise. However, experience has shown that advanced distributed applications executing on top of existing high-performance networks obtain only a very small fraction of the available underlying bandwidth [1, 3,4,5,6,7,10]. It is widely believed that the reason for such poor performance is that the Transmission Control Protocol (TCP) [11], the communication mechanism of choice for most distributed applications, was not designed and is not well suited for a high-bandwidth, high-delay network environment [3,4,5,6,13,14,15]. This issue has led to the development of mechanisms to improve the performance of the TCP protocol itself in this network environment [8,14,16,18], as well as the exploration of user-level techniques that can circumvent some of the problems inherent within the protocol [1,10,13,6a,13a].

In this paper, we also address the fundamental issue of realizing the full power available in a high-bandwidth, high-delay network environment. In particular, we develop and evaluate the performance of FOBS (Fast

Object-Based data transfer System): a very simple user-level object-based[1] communication protocol developed for distributed applications executing in a high-performance, high-delay network environment (i.e. a computational grid). We compare the performance and behavior of FOBS against that of TCP with and without the so-called "Large Window" extensions [6] that have been defined to optimize the performance of TCP in this network environment. This study evaluates the behavior and performance of the two approaches across both short haul and long haul high-performance network connections under a variety of parameter settings and conditions.

While the primary focus of this paper is the comparison of FOBS with TCP, we are also interested in the relative performance and behavior of FOBS and other *user-level* approaches. Given that the use of multiple TCP streams (for a single data flow) is a widely used technique for gridFTP, we also performed a set of experiments comparing the performance of FOBS with PSockets [13]. PSockets attempts to experimentally determine the optimal number of TCP sockets for a given flow, and then transfers the data using this pre-determined number of sockets. It is important to point out however that such comparisons are preliminary, and more research is required before definitive conclusions as to the relative performance of the two protocols can be drawn.

We feel this paper makes three contributions in the area of high-performance distributed computing. First, it outlines a simple user-level object-based communication protocol that is shown to provide excellent performance across both short and long haul connections over high-performance high-delay networks. In particular, FOBS achieved on the order of 90% of the available bandwidth across both the short and long haul connections. In the case of the long haul network, this represents a bandwidth 1.8 times higher than that of an *optimized* TCP implementation. Secondly, this paper provides a detailed evaluation of performance as a function of the various parameters that can be controlled at the user level. Thirdly, this research was conducted using existing "off-the-shelf" connections across the Abilene backbone network without specialized or dedicated network links. This should increase significantly the size of the distributed computing community for which these results will be of interest.

The rest of this paper is organized as follows. In Section 2, we discuss the most closely related research efforts. In Section 3, we provide an overview of FOBS, and discuss the algorithms employed by the data sender and data receiver. In Section 4, we describe the experimental setup for the comparison of FOBS and TCP, and we present the results of this experimentation in Section 5. In Section 6, we provide our initial experiments comparing FOBS and PSockets. We present our conclusions and discuss ongoing and future research in Section 7.

2. Related Work

Obtaining all of the available bandwidth in a high-delay, high-bandwidth network environment is a very important area of current research, and two basic approaches are being pursued. First, there is ongoing research focused on improving the performance of the TCP protocol itself for this network environment. Secondly, there is research aimed at developing techniques at the application level that can circumvent the performance problems associated with TCP. We briefly describe each approach in turn.

As discussed in [13,14,15], the size of the TCP window is the single most important factor in achieving good performance over high-bandwidth, high-delay networks. To keep such "fat" pipes full, the TCP window size should be at least as large as the product of the bandwidth and the round-trip delay. This has led to research in automatically tuning the size of the TCP socket buffers at runtime [16]. Also, it has led to the development of commercial TCP implementations that allow the system administrator to significantly increase the size of the TCP window to achieve better performance [14]. Another area of active research is the use of a Selective Acknowledgement mechanism [8,14,18] rather than the standard cumulative acknowledgement scheme. In this approach, the receiving TCP sends to the sending TCP a Selective Acknowledgement (SACK) packet that specifies exactly those packets that have been received, allowing the sender to retransmit only those segments that are missing.

[1] We define the sense in which it is an object-based data transfer system below.

An excellent source of information, detailing which commercial and experimental versions of TCP support which particular TCP extensions, may be found on the Pittsburgh Supercomputing Center web pages [18].

At the user level, the most common approach to circumventing the performance flaws in TCP is to allocate multiple TCP streams for a given data flow. This is the approach taken by PSockets [13], the grid-FTP protocol [1] (developed by the Globus project [17]) and in [10]. This approach can provide significant performance enhancement for two reasons: First, the limitations on TCP window sizes are on a *per socket* basis, and thus striping the data across multiple sockets provides an *aggregate* TCP buffer size that is closer to the (ideal size) of the bandwidth times round-trip delay. Secondly, this approach essentially circumvents the congestion control mechanisms of TCP. That is, while some TCP streams may be blocked due to the congestion control mechanism, it is likely that some other streams are ready to fire. The larger the number of TCP streams, the lower the probability that *all* such streams will be blocked resulting in a higher probability that *some* TCP stream will always be ready to fire.

The two most closely related user-level approaches are the Reliable Blast UDP Protocol (RUDP [6a]) and SABUL[13a]. In RUDP, all of the data is blasted across the network without any communication between the data sender and receiver. Then, after some timeout period, the receiver sends a list of all missing packets to the sender. The data sender then retransmits all of the lost packets, and this cycle is repeated until all of the data has been successfully transferred. The primary difference between FOBS and RUDP has to do with the types of networks for which the approach is intended. As discussed by the authors, RUDP is designed for high-performance QoS-enabled networks with a very low probability of packet loss. The approach presented here is (still in the process of being) designed for currently available (although non-QoS-enabled) high-performance networks.

SABUL [13a] employs a single UDP stream for data transmission, and a (single) TCP stream for control information related to the state of the data transfer. The primary difference between FOBS and SABUL has to do with the way packet loss is interpreted and how such loss impacts the data-transmission algorithm. In particular, SABUL makes the assumption that packet loss implies congestion, and, similar to TCP, reduces the sending rate to accommodate such perceived congestion. Our approach does *not* make the assumption that packet loss is *necessarily* due to congestion, and further, assumes that some packet loss is inevitable and tolerable when sending packets over wide area networks.

3. FOBS

As noted above, we categorize FOBS as what we term an *object-based* data transfer system. By object-based, we mean that none of the data transferred can be assumed correct until all of the data has been received. This is in contrast to stream-based protocols such as TCP where the application is allowed to access data as it becomes available. This comes at the cost however of requiring that each byte must be delivered to the application in exactly the same order in which it was sent. Streams are kernel-level constructs and are implemented at the kernel level. Objects on the other hand are user-level constructs and are implemented at the user level. This allows knowledge of the characteristics of the data transfer itself to be leveraged to significantly enhance performance.

The fundamental characteristic of an object-based data transfer that is leveraged by FOBS is the assumption that the user-level data buffer spans the entire object to be transferred. For the moment, assume that this is correct and consider how it can be leveraged to enhance performance. In particular, this characteristic allows FOBS to push to the limit the basic concept of the “Large Window” extensions developed for TCP: that is, the window size is essentially infinite since it spans the entire data buffer (albeit at the user level). It also pushes to the limit the idea of selective acknowledgements. Given a pre-allocated receive buffer and constant packet sizes, each data packet in the entire buffer can be numbered. The data receiver can then maintain a very simple data structure with one byte (or even one bit) allocated per data packet, where this data structure tracks the received/not received status of *every* packet to be received. This information can then be sent to the data sending process at a user-defined acknowledgement frequency. Thus the selective acknowledgement window is also in a sense infinite. That is, the data sender is provided with enough information to determine *exactly* those packets, across the *entire* data transfer, that have not yet been received.

FOBS allocates one UDP connection to transfer the data from the sender to the receiver. Another UDP connection is established to send

acknowledgement packets from the receiver to the sender. Additionally, a single TCP connection is opened to send a signal from the receiver to the sender indicating that all of the data has been successfully transferred.

3.1 The Data Sending Algorithm

The data-sending algorithm iterates over three basic phases. In the first phase, the data sender employs some algorithm to determine the number of data packets to be placed onto the network before looking for, and processing if available, an acknowledgement packet. This is referred to as a "batch-sending" operation since all such packets are placed onto the network without interruption (although the **select** system call is used to ensure adequate buffer space for the packet). It is very important to note that after a batch-send operation the data sender *looks for*, but does *not block for*, an acknowledgement packet. It also looks for the completion signal from the receiver.

In the second phase of the algorithm, the data sender looks for, and if available, processes an acknowledgement packet. Processing of an acknowledgement packet entails updating the receive/not received status for each data packet acknowledged, and determining the number of packets that were received by the data receiver between the time it created the previous acknowledgement packet and the time it created the current acknowledgement packet. This information can then be used to determine the number of packets to send in the next batch-send operation. If no acknowledgement packet is available, this information can also be used to determine the number of packets to send in the next batch-send operation. Note that a repeated batch-sending operation with zero packets is logically equivalent to blocking on an acknowledgement.

In the third phase of the algorithm, the data sender executes some user-defined algorithm to choose the next packet, out of *all* unacknowledged packets, to be placed onto the network.

3.1.1 Parameters for the Data Sender

The first parameter studied was the number of packets to be sent in the next batch-send operation. Intuitively, one would expect that the data sender should check for an acknowledgement packet on a very frequent basis, thus limiting the number of packets to be placed onto the network in a given batch-send operation. Our experimental results supported this intuition, finding that two packets per batch-send operation provided the best performance. We therefore used this number in all subsequent experimentation

We also performed extensive experimentation to determine which particular packet, out of all unacknowledged packets, should next be placed onto the network. We tried several algorithms, and, in the end, it became quite clear that the best approach (by far) was to treat the data as a circular buffer. That is, the algorithm never went back to *re-transmit* a packet that was not yet acknowledged, if there were *any* packets that had not yet been sent for the first time. Similarly, a given packet was re-transmitted for the **n+1st** time *only* if all other unacknowledged packets had been re-transmitted **n** times.

As can be seen, the algorithm executed by the sender is very greedy, continuing to transmit (or re-transmit) packets (without blocking) until it receives the completion signal from the data receiver specifying that all data has been successfully received. Thus a reasonable question to ask is how wasteful of network resources is this approach.

To answer this question, we maintained a count of the total number of packets placed on the network by the data sender. We define wasted resources as the *total* number of packets sent, minus the number of packets that must be transferred, divided by the number of packets that must be transferred.

3.2 The Data Receiving Algorithm

The data receiver basically polls the network for new packets, and, when a packet becomes available, incorporates it into its proper place within the data buffer as determined by its sequence number.

The most important parameter with respect to the data receiver is the number of *new* packets received before generating and sending an acknowledgement packet. The frequency with which the data receiver sends acknowledgement packets essentially determines the level of synchronization between the two processes. A small value (and thus a high level of synchronization) implies that the data receiver must frequently stop pulling packets off of the network to create and send acknowledgement packets. Given that the algorithm is UDP-based, those packets missed while creating and sending

an acknowledgement will, in all likely-hood, be lost. A very high value, and thus a very low level of synchronization, results in both the data sender and data receiver spending virtually all of their time placing packets on, and reading packets off, of the network. We show the performance of the algorithm as a function of the acknowledgement frequency below.

4. Experimental Design

We investigated the performance characteristics of (reasonably) large-scale data transfers between various sites connected by the Abilene backbone network. We investigated the performance of TCP (with and without the Large Window Extensions), and compared this with the performance and behavior of FOBS. One connection tested was between Argonne National Laboratory (ANL) and the Laboratory for Computational Science and Engineering (LCSE) at the University of Minnesota. The slowest link in this path was 100 Mb/Sec (from the desktop computer to the external router at ANL). Also, we performed a set of experiments between ANL and the Center for Advanced Computing Research (CACR) at the California Institute of Technology. ANL is connected to both of these sites across Abilene. The endpoints at both ANL and LCSE were Intel Pentium3-based PCs running Windows 2000 and using the Winsock2 API.

We experimented with two endpoints at CACR. One was an SGI Origin200 with two 225Mhz MIPS R10000 processors, one Gigabyte of RAM, and a 100Mb/Sec interface card. The other was a HP V2500 system with 64 CPUs (440Mhz PA-8500 64-bit RISC processors), a 100Mb/Sec external interface card, running the HP-UX 11.10 operating system. The HP-UX 11.10 operating system automatically provides window scaling and timestamps when the user requests a socket buffer size greater than 64K. The SGI Origin200 requires kernel-level access to increase the TCP buffer size (which we did not have). We also conducted experiments between an SGI Origin2000 (with 49 processors running IRIX 6.5) at NCSA and the Windows box at LCSE. We were interested in this connection because both endpoints had a Gigabit Ethernet network interface card with an OC-12 connection to the Abilene backbone network.

The round-trip delay between ANL and LCSE was measured (using traceroute) to be on the order of 26 milliseconds, and we (loosely) categorized this as a *short haul* network. The round-trip delay between ANL and CACR was on the order of 65 milliseconds, which we (again loosely) categorized as a *long haul* network.

The transmitted data size for the experiments was 40 MB, where the data was divided into equal fixed-sized packets of 1024 bytes (which was less than the MTU for all network links considered). The metric of interest was the percentage of the maximum available bandwidth obtained by each approach.

5. Experimental Results

Figure 1 shows the performance of FOBS across the short haul connection (between ANL and LCSE), and the long haul connection (between ANL and CACR). As noted, performance was measured as the percentage of the maximum available bandwidth obtained by FOBS (where the 100 Mb/Sec interface card was the limiting factor). Performance is shown as a function of the number of packets received before triggering an acknowledgement packet. As can be seen, FOBS provides excellent performance across both the long and short haul connections, achieving approximately 90% of the available bandwidth across both network connections.

Figure 2 provides a simple measurement of the amount of network resources wasted due to the greedy nature of the algorithm. This was calculated as the total number of packets placed on the network, minus the total number of packets required to complete the data transfer, divided by the total number of packets required. This is also presented as a function of the acknowledgement frequency.

As noted, the limiting factor in these experiments was the 100 Mb/Sec network interface card of the desktop machine at ANL. We also performed a series of tests between an SGI Origin2000 at NCSA, and the Intel Pentium3 Windows 2000 box at LCSE (described in Section3), both of which had Gigabit Ethernet network connections with an OC-12 (622 Mb/Sec) data link to the Abilene backbone network. The SGI Origin2000 had 48 R1000 processors with a total memory of 14 GB running IRIX 6.5. We looked at the percentage of the maximum available bandwidth obtained as a function of the UDP packet size. These results are shown in Figure 3. As can be seen, the size of the data packet makes a tremendous difference in performance, and the performance of FOBS

peaked out at approximately 52% of the maximum available bandwidth (40 MB/Sec).

5.1. Performance of TCP

Table 1 shows the performance of TCP across the short and long haul connections. As discussed in Section 3, we experimented with two endpoints at CACR: an SGI Origin200 (that requires kernel-level access to increase the TCP window size), and the HP V2500 system that utomatically provides window scaling when the user requests a buffer size greater than 64KB. As can be seen, the results obtained using the Windows 2000 TCP implementation (across the short haul network) were quite impressive, providing approximately the same performance as that of FOBS. Two factors combined to make such performance possible: First, both endpoints provided automated support for the Large Window extensions to TCP. Secondly, there was virtually no contention in the network and thus the congestion control mechanisms of TCP were not triggered.

As can be seen however, the performance of TCP drops dramatically over the long haul connections. The performance was significantly better when both endpoints provided automatic support for the Large Window Extensions to TCP, achieving on the order of 50% of the available bandwidth. Without such support, this performance decreased to approximately 10% of the available bandwidth. The reason for this dramatic drop in performance (even with the Large Window Extensions enabled) was most likely caused by to the presence of *some* contention in the network, which triggered TCP's very aggressive congestion control mechanisms.

6. Comparison with PSockets

We were interested in comparing the performance and behavior of FOBS with other user-level approaches currently under development. We chose PSockets for this comparison primarily because multiple TCP streams are often employed for gridFTP applications. A secondary (but important) factor was that the PSockets library was very easy to install, build, and use. We conducted this set of experiments between the SGI Origin2000 housed at NCSA and the HP VV2500 system located at CACR (both are described in detail in Section 4).

The results are provided in Table 2. As can be seen, FOBS was able to obtain 76% of the available bandwidth while Psockets obtained 56% of this maximum. The results obtained for PSockets are somewhat less than those reported elsewhere [13]. Similarly, the performance of FOBS was somewhat less than that observed across the other connections tested. It seems reasonable to assume that this reduced performance (in both cases) was a function of increased contention for network resources.

7. Conclusions and Future Research

In this paper, we have described a very simple, user-level communication protocol designed for a high-bandwidth, high-delay network environment. We have shown that the algorithm performs quite well, achieving approximately 90% of the available bandwidth on both short and long connections over the Abilene backbone network. Also, we have shown that the algorithm can achieve on the order of 50% of the available bandwidth when the communication endpoints have Gigabit Ethernet cards and an OC-12 connection to the Abilene backbone network. Further, we demonstrated that the additional load placed on the network due to the greedy nature of the algorithm is quite reasonable, representing approximately 3% of the total data transferred.

Clearly there is much research left to be done. Perhaps most importantly, FOBS does not yet provide congestion control. This is a reasonable first cut given that the primary issue (at least currently) is one of making efficient use of high-performance networks rather than one of congestion in such networks. However, some form of congestion control is needed before the algorithm can become generally used.

There are two ways of addressing this issue that are being explored. First, we are looking at modifying FOBS such that it switches to a high-performance TCP algorithm when congestion in the network is detected *and* when such congestion is determined to be of more than temporary duration. Then, when the congestion appears to have dissipated, FOBS could switch back to the greedy implementation of the algorithm. Alternatively, we are investigating mechanisms to decrease the greediness of FOBS when congestion in the network is detected (and is of sufficient duration).

In conclusion, it is important to point out the difficulties encountered in trying to pursue research of this nature. In particular, since

network conditions are constantly changing it is very difficult to find windows of time when two or more approaches can be compared in a meaningful way. For this reason, we are also engaged in the development of simulation models that can be used to compare the various algorithms under similar (albeit simulated) loads and traffic mixes.

References

[1] Allcock, B. Bester, Bresnahan, J., Chervenak, A., Foster, I., Kesselman, C. Meder, S., Nefedova, V., Quesnet, D., and S. Tuecke. *Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing*. Preprint ANL/MCS-P871-0201, Feb. 2001.

[2] Allman, M., Paxson, V., and W.Stevens. *TCP Congestion Control. RFC 2581*, April 1999.

[3] Feng, W. and P. Tinnakornsrisuphap. *The Failure of TCP in High-Performance Computational Grids.* In the Proceedings of the Super Computing 2000 (SC2000).

[4] Hobby, R. *Internet2 End-to-End Performance Initiative (or Fat Pipes Are Not Enough).* URL: http//www.internet2.org.

[5] Irwin, B. and M. Mathis. *Web100: Facilitating High-Performance Network Use.* White Paper for the Internet2 End-to-End Performance Initiative. URL:http://www.internet2.edu/e2epi/web02/p_web100.shtml

[6] Jacobson, V., Braden, R., and D. Borman. *TCP Extensions for high performance.* RFC 1323, May 1992.

[6a] Leigh, J. et al. Adaptive Networking for Tele-Immersion. In: *Proceedings of the Immersive Projection Technology/Eurographics Virtual Environments Workshop (IPT/EGVE)*, Stuttgart, Germany, 05/16/01-05/18/01.

[7] MacDonald and W. Barkley. *Microsoft Windows 2000 TCP/IP Implementation Details*. White Paper, May 2000.

[8] Mathis, M., Mahdavi, J., Floyd, S. and A. Romanow. *TCP Selective Acknowledgement Options. RFC 2018.*

[9] Mogul, J. and S. Deering, "Path MTU Discovery", RFC 1191, November 1990.

[10] Ostermann, S., Allman, M., and H. Kruse. *An Application-Level solution to TCP's Satellite Inefficiencies.* Workshop on Satellite-based Information Services (WOSBIS), November, 1996.

[11] J. Postel, *Transmission Control Protocol*, RFC793, September 1981.

[12] Semke, J., Jamshid Mahdavi, J., and M. Mathis, *Automatic TCP Buffer Tuning*, Computer Communications Review, a publication of ACM SIGCOMM, volume 28, number 4, October 1998].

[13] Sivakumar, H., Bailey, S., and R. Grossman. *PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks*. In Proceedings of Super Computing 2000 (SC2000).

[13a] Sivakumar, H., Mazzucco, M., Zhang, Q., and R. Grossman. Simple Available Bandwidth Utilization Library for High Speed Wide Area Networks. Submitted to *Journal of SumperComputing*

[14] URL: http://www.psc.edu/networking/perf_tune.html#intro. Enabling High Performance Data Transfers on Hosts: (Notes for Users and System Administrators). Pittsburgh Supercomputing Center.

[15] URL: http://dast.nlanr.net/Articles/GettingStarted/TCP_window_size.html. National Laboratory for Advanced Networking Research.

[16] URL: http://dast.nlanr.net/Projects/Autobuf_v1.0/autotcp.html. Automatic TCP Window Tuning and Applications. National Laboratory for Advanced Networking Research.

[17] URL: http://www.globus.org. The Globus Project.

[18] URL: http://www.psc.edu/networking/all_sack.html. List of sack implementations, Pittsburgh Supercomputing Center.

[19] URL: http://www.internet2.org. The Internet2 project.

[20] URL: http://www.internet2.edu/abilene. The Abilene backbone network.

[21] URL: http://www.vbns.net. The Very High Performance Backbone Network Service.

Figures

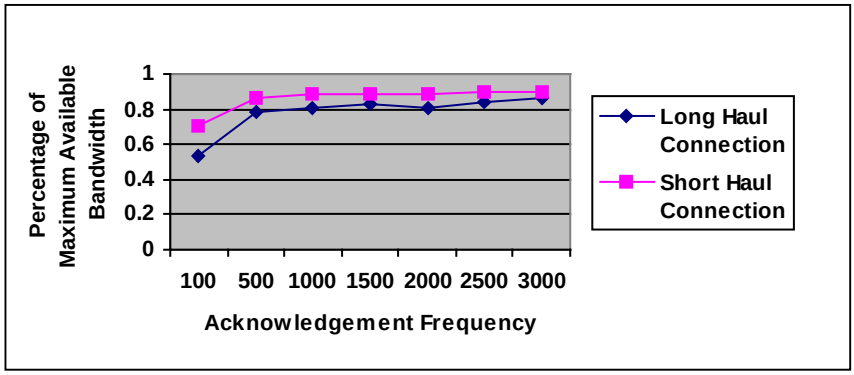

Figure 1. This figure shows the performance of FOBS across both short and long haul high-performance network connections. Performance is measured as the percentage of maximum bandwidth achieved and is shown as a function of number of packets received before triggering an acknowledgement packet.

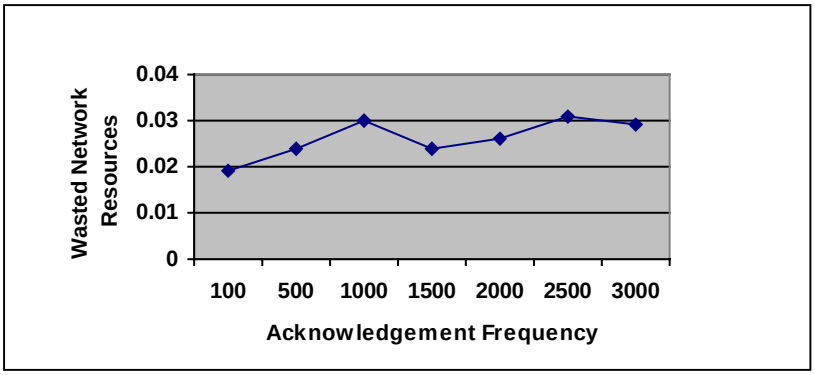

Figure 2. This figure shows the amount of wasted network resources as a function of the number of packets received before triggering an acknowledgment packet. The amount of waste was calculated as the total number of packets required to complete the transfer divided by the total number of packets actually sent.

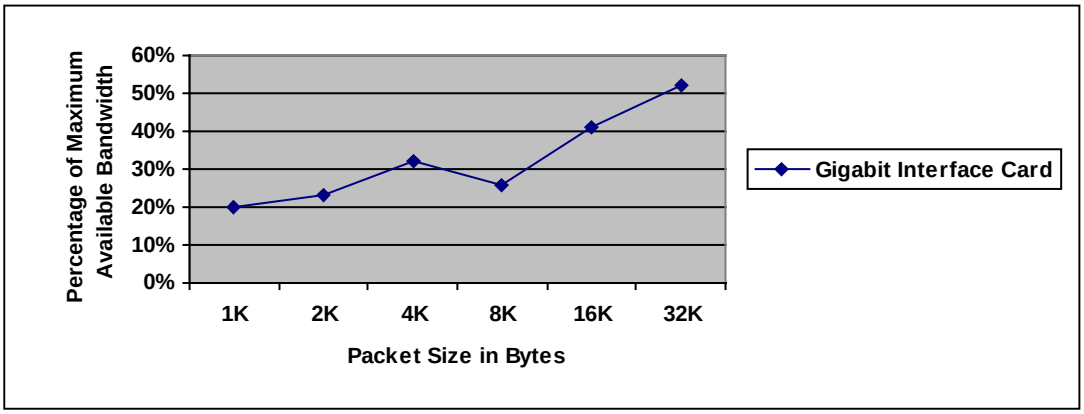

Figure 3. This figure shows the percentage of maximum bandwidth obtained over a short haul connection where the communication endpoints both had Gigabit Ethernet network interface cards and an OC-12 connection to the Abilene backbone network. Performance is shown as a function of the UDP packet size.

Network Connection	Percentage of the Maximum Available Bandwidth
Short Haul with LWE	86%
Long Haul with LWE	51%
Long Haul without LWE	11%

Table 1. This table shows the percentage of the maximum bandwidth obtained by TCP with and without the Large Window extensions.

	PSockets	FOBS
Percentage of Maximum Bandwidth Obtained	56%	76%
Percentage of Wasted Network Resources	—	2%
Optimal Number of Parallel Sockets	20	—

Table 2. This table compares the performance of FOBS with that of PSockets across one high-performance network connection.