

INCREASING TCP THROUGHPUT BY USING AN EXTENDED
ACKNOWLEDGEMENT INTERVAL

A thesis presented to

The Faculty of

The College of Arts and Sciences of Ohio University

In Partial Fulfillment
of the Requirements for the Degree

Master of Science

Stacy R. Johnson

June 1995

INCREASING TCP THROUGHPUT BY USING AN EXTENDED
ACKNOWLEDGEMENT INTERVAL

BY

STACY R. JOHNSON

This thesis has been approved
for the Department of Mathematics
and the College of Arts and Sciences by

Shawn D. Ostermann
Assistant Professor of Computer Science

Harold Molineu
Dean, College of Arts and Sciences

To my family

ABSTRACT

Johnson, Stacy R.. M.S., Ohio University, June 1995. Increasing TCP Throughput by Using an Extended Acknowledgement Interval. Major Professor: Dr. Shawn D. Ostermann.

The Transmission Control Protocol (TCP) is a widely used network protocol that is usually layered over the Internet Protocol (IP). IP is not reliable; it does not insure data delivery. However, TCP is reliable, and insures data delivery through the use of acknowledgements and retransmissions.

The receiving side of a TCP connection currently acknowledges at least every other data block (segment) that it receives. Some of these acknowledgements may be superfluous. A large portion of a TCP connection's cost is the overhead of processing a new packet as it arrives. Consequently, it is possible that extending the acknowledgement interval may increase throughput by requiring less acknowledgement overhead per connection.

We implemented extended acknowledgement intervals in UNIX 4.4 BSD compatible network code. We compiled the code on a Sun IPC workstation running SunOS.

To test our hypothesis, we conducted throughput tests using machines on the same local network, a near network, and a distant network. Analysis of the data revealed that extending the acknowledgement interval does increase throughput by a statistically significant amount in the local and near network cases. In particular, in the local case the throughput was increased by as much as 4 percent in some cases. It is probable that congestion on the Internet and the resulting packet loss prevented the overall increase in throughput on the non-local connections.

ACKNOWLEDGMENTS

Thanks to my family and my friends; your support over the past year is greatly appreciated. Special thanks to my fellow members of Ohio University's Internetworking Research Group for the lively discussions. Also, thanks to Dr. Ostermann for your ideas, patience, and for introducing me to the wide world of networking.

DISCARD THIS PAGE

TABLE OF CONTENTS

	Page
ABSTRACT	4
LIST OF TABLES	8
LIST OF FIGURES	9
1. INTRODUCTION	10
1.1 TCP/IP Overview	10
1.2 Acknowledgements in TCP	12
1.2.1 TCP Processing Overhead	13
1.2.2 Current TCP Acknowledgement Implementation	14
1.2.3 Timing TCP Segments	14
1.2.4 Delayed Acknowledgements and the Round Trip Time	15
1.3 Window Size and Throughput in TCP	16
1.4 Extended Acknowledgement Intervals	18
2. IMPLEMENTATION	19
2.1 Environment	19
2.2 Implementation Details	19
2.2.1 tcp_var.h: Tracking Fields	19
2.2.2 tcp_input.c	21
2.2.3 tcp_output.c	22
2.2.4 tcp_subr.c	23
2.2.5 tcp_timer.c	23

	Page
3. EXPERIMENTAL RESULTS	25
3.0.6 Statistical Analysis	25
3.1 Local Network	27
3.1.1 Gathering data	28
3.1.2 Experiments with a 4k Window Size	28
3.1.3 Experiments with a 51k Window Size	33
3.1.4 Experiments with a 32k Window Size	35
3.2 Near Network	37
3.2.1 Explanation of Results	39
3.3 Distant Network	41
3.3.1 Explanation of Results	43
4. CONCLUSIONS	44
4.1 Overview	44
4.2 Conclusions	44
4.3 Recommendations for Further Research	45
BIBLIOGRAPHY	47
 APPENDIX	
A. HOW TO READ A TCP TIME SEQUENCE GRAPH	49

LIST OF TABLES

Table	Page
1.1 The Effect of TCP Window Size on Throughput	17
2.1 tcp_input.c Tracking Variables	22
2.2 tcp_output.c Tracking Variables	23

APPENDIX

Table

LIST OF FIGURES

Figure	Page
1.1 Protocol Layering on the Internet	11
1.2 TCP Header Format	13
2.1 TCP Research Variable Structure	20
2.2 Extending the Acknowledgement Interval	21
3.1 Raw Throughput Curve	26
3.2 Adjusted Throughput Curve	27
3.3 Local Network Throughput - 4k Buffer Size	29
3.4 Average Number of Acknowledgements per Connection	31
3.5 4k Window Size Leads to Ack-Every-Other Conditions	32
3.6 Local Network Throughput - 51k Buffer Size	34
3.7 Effect of Buffer Size on Local Network Throughput	35
3.8 Local Network Throughput - 32k Buffer Size	36
3.9 Extended Acknowledgement Interval of 6 - 32k Window Size	38
3.10 Acknowledgement Interval of 2 (Default) - 32k Window Size	39
3.11 Near Network Throughput	40
3.12 Distant Network Throughput	42
A.1 Example TCP Time Sequence Graph	50

1. INTRODUCTION

1.1 TCP/IP Overview

A network protocol is a set of rules that computers use when exchanging data over a computer network. TCP and IP are the two protocols that form the basis of sharing information over the Internet. A protocol is *reliable* if it guarantees data delivery. An *unreliable* protocol does not guarantee data delivery. A protocol is *connectionless* if each packet sent over the protocol is treated independently [Com88]. With a connectionless protocol, each packet may take a different path to reach its destination, and the packets may arrive in any order.

Data is split into pieces before it is sent over a network. Before transmitting the data, the protocol adds a header that includes the source and destination of the data. On an Ethernet, the resulting unit is called a physical *frame* [Com88]. At a higher protocol level, each unit may be called a *datagram* or a *segment*.

Network protocols are often layered. A complete packet including the header and data from one protocol may be contained, or *encapsulated*, within the data section of another protocol. A high-level protocol may be encapsulated within a low-level protocol, which may itself be encapsulated within another protocol. Each protocol layer adds useful features to the one below it.

The *Internet Protocol* (IP) is an unreliable, connectionless protocol that delivers data between computers over the Internet. Each unit sent using IP is called a datagram.

The *Transmission Control Protocol* (TCP) is a reliable, connection-oriented [Com88] protocol that delivers data between applications. TCP can be used to deliver data between applications on the same computer, or to different computers over an internet. Although TCP is a general purpose protocol which can be used with several delivery systems, on the Internet its data units, called *segments*, are encapsulated within IP datagrams [Com88] as shown in figure 1.1. Even though TCP is encapsulated within the connectionless IP protocol, it is connection-oriented [Com88] because hosts must negotiate to open and close a TCP connection.

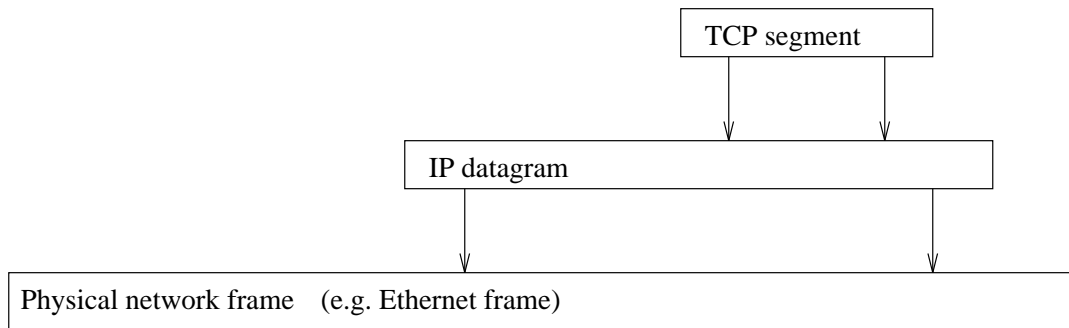


Figure 1.1 Protocol Layering on the Internet

When using TCP on the Internet, data is encapsulated within a TCP segment, an IP datagram, a physical frame, and then transmitted.

1.2 Acknowledgements in TCP

As mentioned previously, IP is an unreliable protocol. IP has a “best-effort” delivery scheme, but does not guarantee the delivery of any data. To compensate for this, TCP is reliable; it guarantees the delivery of data through the use of acknowledgements and retransmissions..

Each byte of data that is sent through a TCP connection is acknowledged using a *positive acknowledgement*, which means that the receiving TCP acknowledges data that it receives¹. If the receiving TCP does not acknowledge data within a certain time limit (see section 1.2.3) the sending TCP will retransmit the data. TCP acknowledgements are also cumulative; a host acknowledging receipt of the n th byte is also implicitly acknowledging the receipt of all bytes previous to n .

Each side of a TCP connection chooses an initial *sequence number* for the first byte of data that TCP sends over the network. Subsequent data bytes are numbered sequentially starting at this number. The receiving TCP must acknowledge receipt of every data byte sent through a TCP connection, at least cumulatively. The receiving TCP acknowledges the data by sending an acknowledgement (ACK) which contains the sequence number of the next byte it expects to receive. For example, if a segment arrives containing bytes with the sequence numbers 100 - 200, the receiving TCP can acknowledge all of the bytes in the segment by sending an acknowledgement containing the sequence number 201.

TCP connections are *full-duplex*, which means that data can flow in both directions simultaneously [Com88]. The fields that are used for acknowledgements are contained in the header of the TCP segment (see figure 1.2), which allows TCP to send an acknowledgement and data in the same segment. This is known as *piggybacking* an acknowledgement on a data segment.

¹In contrast, some protocols use a *negative acknowledgement*; the receiver will send a message if it does not receive expected data.

0	4	10	16	24	31
Source Port			Destination Port		
Sequence Number					
Acknowledgement Number					
Data Offset	Reserved	Flags	Window		
Checksum			Urgent Pointer		
Options				Padding	
Data (variable length)					

Figure 1.2 TCP Header Format [Pos81]

There is space reserved for an acknowledgement in each TCP segment. A host may send data and an acknowledgement in the same packet by setting the acknowledgement flag and filling the acknowledgement field with the 32 bit sequence number of the next byte it expects to receive.

1.2.1 TCP Processing Overhead

A simple version of TCP acknowledges every segment sent. Thus, for every data segment sent over the network, there is also an acknowledgement segment. Most of the cost of maintaining a TCP connection, in terms of computer resources, is due to the overhead of dealing with incoming packets [Cla82]. Recall that TCP acknowledgements are cumulative; those sent near each other may be redundant. Eliminating some of the redundant acknowledgements will help to alleviate connection overhead.

1.2.2 Current TCP Acknowledgement Implementation

The current specification of TCP [Bra89] recommends *delayed acknowledgements*. Specifically, after receiving a segment, the receiving TCP may delay sending the acknowledgement for up to 500 ms. BSD-based implementations of TCP will delay sending the acknowledgement for up to 200 ms [WS95]. If the host has outgoing data for the same connection, TCP can piggyback the acknowledgement on the next data segment. This is useful for cutting down on the number of segments in interactive connections such as a telnet connection, where each character typed is immediately echoed back to the sender. Furthermore, according to the standard, the receiving TCP must send an acknowledgement for at least every second segment [Bra89].

1.2.3 Timing TCP Segments

As mentioned previously, the receiving TCP uses positive acknowledgements to signal receipt of data segments. If a data segment is lost or damaged, the receiving TCP will not acknowledge it. In this case, the sending TCP needs a mechanism to recognize when to resend a segment. The sending TCP maintains a *retransmission timeout* value (RTO), which is the amount of time that TCP will wait before re-sending a segment [Ste94].

The *round trip time* (RTT) of a segment is the elapsed time between sending a segment and receiving an acknowledgement that covers the segment. The RTO value is based on the round trip times of TCP segments flowing over the given connection. The RTT of segments flowing over a connection can vary widely [Jac88], so the RTT of individual segments are combined through a smoothing algorithm [Jac88] to provide a better estimate of the RTT for subsequent segments.

TCP has two timers running at all times: a 200 ms *fast timer* and a 500 ms *slow timer*. TCP measures the RTT of a segment in slow timer ticks, a granularity of 500

ms [WS95]. The TCP Vegas research project [BOP94] claims to have gotten higher throughput by using UNIX timestamps² with a millisecond granularity, as opposed to the slow timer tick.

TCP times only one segment at a time [Ste94]. Consequently, TCP only times 1 segment per every n segments, where n is the number of segments in transit at a given time. The sampling of the RTT is dependent on the number of segments in transit simultaneously [JBB92]. Several suggestions have been made for improving the accuracy of the RTT measurements. For example, if TCP timed every segment, it could apply the RTT to the last segment covered by the acknowledgement. This would increase the RTT sampling rate, and would result in a more accurate RTT. Such a scheme is proposed in TCP Vegas [BOP94]. TCP's timestamp option [JBB92] offers another way to get a more accurate RTT. When using this option, TCP includes a timestamp in each segment that is sent. The receiver echoes this timestamp in the acknowledgement segment. The sender is able to calculate a RTT for each acknowledgement it receives [JBB92]. Currently, TCP can only calculate a RTT for an acknowledgement of a timed segment (one segment per window) [Ste94].

1.2.4 Delayed Acknowledgements and the Round Trip Time

Delayed acknowledgements affect the RTT because of the way that TCP currently times the segments (see section 1.2.3). Assume that the sending TCP sends the segment it is timing, segment n , followed by segment $n + 1$. If the receiving TCP delays acknowledgement of segment n , then the sending TCP will calculate a longer RTT for the segment n when it receives the acknowledgement of segment $n + 1$ [Ste94].

TCP's timestamp option may be used in conjunction with delayed acknowledgements. If delayed acknowledgements are being used, the receiving TCP echoes the

²A UNIX timestamp is an integer which represents the present time as the number of seconds since January 1, 1970.

earliest timestamp it received that it has not yet acknowledged. The sending TCP then automatically takes into account the time for the delayed acknowledgements [JBB92].

When using delayed acknowledgements as we are in this research, a longer (incorrect) RTT measurement is vital. If the RTT is accurate, TCP may construe the longer RTT caused by the delayed acknowledgements to mean lost segments, and start retransmitting segments unnecessarily [Bra89].

1.3 Window Size and Throughput in TCP

Each side of a TCP connection has a buffer for incoming data. The amount of buffer space available is called the *window size*. Data sent through the connection is held in the receiving TCP's buffers until it can be delivered to the application reading from the connection. The available window size will vary throughout the duration of the connection, depending on how much data has arrived at the host and is still waiting to be processed.

If you visualize the connection between two hosts as a pipeline, then the size of the buffers on both sides of the connection limit how much information can be in the pipeline at one time. The amount of information in the pipeline must never exceed the receiving TCP's buffer space. TCP will attempt to fill the connection between the hosts with as much data as possible. In general, as the buffer size grows, so does the amount of data that can be in transit at a given time.

Throughput is a measure of the amount of data flowing through a connection per unit time. In general, one desires to maximize the throughput of a given connection. The TCP window size affects throughput by limiting the amount of data that can be in transit at a given time. For example, if the window size of the receiving host is

small, then the sending host will have to pause periodically throughout the duration of the connection to allow the window to open again [Ste94].

If the sending TCP manages to fill the receiving TCP’s buffers, the receiving TCP will send a *zero window advertisement* which causes the sending TCP to stop transmitting data over the connection [Ste94]. The sending TCP will set a persist timer; if it has not received a window update when the timer expires, it will send a message to the receiving TCP to see if a window update has been lost [Ste94]. During the time that the sending TCP waits for the receiving TCP to empty information from its buffers, the pipeline between the hosts may empty out. Once buffer space becomes available, it will take time to refill the pipe with data and have the connection running at “full capacity” again. Table 1.1 shows the theoretical maximum throughput for various window sizes, as calculated by the bandwidth-delay product equation [Ste94].

Buffer Size	Round Trip Time (ms)				
	local (2.5 ms)	cross-campus (4.5 ms)	Kent State (60 ms)	U of Kentucky (120 ms)	Sunsite CA (270 ms)
4k	1.56	0.87	0.07	0.03	0.01
16k	6.25	3.47	0.26	0.13	0.06
32k	12.5	6.94	0.52	0.26	0.12
64k	25.0	13.89	1.04	0.52	0.23

Table 1.1 The Effect of TCP Window Size on Throughput

For each host/RTT and buffer size listed, throughput is shown in MB/s. The maximum possible throughput increases as the TCP window size (buffer size) increases.

1.4 Extended Acknowledgement Intervals

Based on our observations of TCP, we hypothesize that extended acknowledgement intervals may allow for greater throughput on a TCP connection. TCP currently requires a host to acknowledge at least every other segment [Bra89]. This is an acknowledgement interval of two; two segments may be received before an acknowledgement must be sent. It is possible that extending the acknowledgement interval may increase throughput by requiring less acknowledgement overhead per connection.

The research presented in the rest of this paper tests the effects of extended acknowledgement intervals on the throughput of local and non-local TCP connections. Chapter 2 focuses on the necessary operating system changes to implement extended acknowledgement intervals. Chapter 3 describes the experiments performed with the modified TCP. Chapter 4 examines the results of the experiments and gives suggestions for further research.

2. IMPLEMENTATION

2.1 Environment

We compiled an experimental operating system (kernel) on a Sun IPC workstation (netipc2) running SunOS 4.1.2, from Sun Microsystems. We made changes to 4.4 BSD compatible public domain networking code, and compiled the networking code into the SunOS 4.1.2 operating system. The workstation is attached to a local Ethernet network.

2.2 Implementation Details

Few changes are needed to implement extended acknowledgement intervals in TCP. Most of the changes that we made track the effects of the extended acknowledgement intervals. The affected files are: `tcp_input.c`, `tcp_output.c`, `tcp_subr.c`, `tcp_timer.c`, and `tcp_var.h`. The following sections describe the changes made to each of the files.

2.2.1 `tcp_var.h`: Tracking Fields

Each TCP connection is associated with a TCP control block that holds information about the connection. TCP keeps track of several statistics. For example, TCP

tracks the number of bytes sent, the number of bytes received, and the number of duplicate packets received. However, TCP does not track these on a per-connection basis. We added a new structure that contains per-connection tracking data (figure 2.1). The TCP control block has a pointer to this structure. We placed code to update these tracking variables next to where the SunOS tracking variables were maintained in the basic TCP code.

The new version of TCP also keeps track of the number of times the extended interval code is called during the lifetime of each TCP connection (variables `t_inc`, `t_ackdone`, and `t_timeack` in figure 2.1).

```

#ifdef TCPEXP
#define MT_TCPE          20          /* mbuf type for our experimental variables */
                                  /* this should be in mbuf.h */
/* This structure holds the tcp experimental fields/statistics. */
/* A pointer to this structure is located in the TCP control block. */
struct tcpe {
    u_long    t_acknowcnt;          /* countdown to setting acknow flag */
    u_long    t_inc;               /* num times acknowcnt increased */
    u_long    t_ackdone;          /* num times acknow reached limit */
    u_long    t_timeack;          /* num times timed out and acked */
    u_long    t_sndbyte;          /* data bytes sent */
    u_long    t_sndrexmitpack;    /* retransmitted packets */
    u_long    t_sndrexmitbyte;    /* data bytes retransmitted */
    u_long    t_sndacks;          /* ack-only packets sent */
    u_long    t_rcvbyte;          /* bytes received in sequence */
    u_long    t_rcvdupbyte;       /* duplicate-only bytes received */
    u_long    t_rcvpartdupbyte;   /* dup. bytes in part-dup. packets */
    u_long    t_rcvackpack;       /* acks received */
    u_long    t_rcvackbyte;       /* bytes acked by rcvd acks */
    u_long    t_rcvdupack;        /* duplicate ack packets received */
};
#endif

```

Figure 2.1 TCP Research Variable Structure

The experimental variables structure houses the variables used in this research. The TCP control block structure has a pointer leading to this structure.

2.2.2 tcp_input.c

The most extensively altered file in the implementation is `tcp_input.c`. The changes to `tcp_input` comprise the bulk of the extended acknowledgement interval implementation. The heart of the code is shown in figure 2.2.

```

if (tcp_nodelack == 0) {
    (tp->t_exp->t_acknowcnt)++;
    (tp->t_exp->t_inc)++;
    if (tp->t_exp->t_acknowcnt == TCPE_ACKNOWCNT) {
        tp->t_flags |= TF_ACKNOW;
        tp->t_exp->t_acknowcnt = 0;
        (tp->t_exp->t_ackdone)++;
    } else
        tp->t_flags |= TF_DELACK;
} else
    tp->t_flags |= TF_ACKNOW;

```

Figure 2.2 Extending the Acknowledgement Interval

This piece of code in `tcp_input.c` checks the number of “outstanding” segments to see if the extended acknowledgement interval limit has been reached. If the limit has been reached, it sets a flag to acknowledge the segment.

This piece of code is placed in the *dodata* section of `tcp_input`, which processes an incoming segment and arranges for an acknowledgment of the segment, if necessary.

If delayed acknowledgements are not enabled (they are enabled by default), a flag is set to acknowledge the packet. If delayed acknowledgements are enabled, then three things occur. First, the field that counts the number of outstanding acknowledgements (`t_acknowcnt`) is incremented. Second, the `t_inc` is incremented to show that the outstanding acknowledgements field was incremented.. Third, TCP decides whether the current segment needs to be acknowledged immediately by comparing the number of outstanding acknowledgements to the preset limit.

If the number of outstanding acknowledgements reaches the preset limit (`TCPE_ACKNOWCNT`), then the flag is set to acknowledge the segment immediately. The outstanding acknowledgements field (`t_acknowcnt`) is reset, and the tracking variable

`t_ackdone` is incremented to show that the acknowledgement was sent because the number of outstanding acknowledgements met the preset limit.

The code that sets the initial threshold value for outstanding acknowledgements (`TCPE_ACKNOWCNT`) is also contained in `tcp_input.c`. Setting the interval to 2 evokes the “ack-every-other” behavior of the standard 4.4 BSD TCP code.

The tracking variables altered by routines in `tcp_input.c` are listed in Table 2.1

Tracking Variables - Receiving	
Name	Description
<code>t_rcvackbyte</code>	bytes acked by the acknowledgement packets received
<code>t_rcvackpack</code>	acknowledgement packets received
<code>t_rcvbyte</code>	bytes received
<code>t_rcvdupack</code>	duplicate acknowledgement packets received
<code>t_rcvdupbyte</code>	duplicate bytes received
<code>t_rcvpartdupbyte</code>	duplicate bytes in partially duplicate packets

Table 2.1 `tcp_input.c` Tracking Variables

The tracking variables that are incremented by routines in the `tcp_input.c` file.

2.2.3 `tcp_output.c`

All changes to `tcp_output.c` involve tracking the effects of the extended acknowledgement intervals on TCP’s sending side. The tracking variables that are modified in `tcp_output.c` are listed in 2.2.

Tracking Variables - Sending	
Name	Description
t_sndacks	ack-only (non-piggybacked) acknowledgements sent
t_sndbyte	data bytes sent
t_sndremitbyte	retransmitted bytes
t_sndremitpack	retransmitted packets

Table 2.2 tcp_output.c Tracking Variables

The tracking variables that are incremented by routines in the tcp_output.c file.

2.2.4 tcp_subr.c

When a new socket is created for a TCP connection, the `tcp_newtcpcb` subroutine is called to allocate and initialize a new TCP control block [WS95]. The code that we added to `tcp_newtcpcb` allocates and initializes a structure to hold the extra variables needed for this implementation. TCP accesses these variables through a pointer inside the TCP control block structure. The control block cannot hold the information itself due to the size restrictions of the BSD mbuf structure [LMKQ89].

A closing TCP connection calls the `tcp_close` function [WS95]. The code added to this function prints the connection's tracking field variables to the system log. The `tcp_close` function also deallocates the memory used for the tracking fields structure.

2.2.5 tcp_timer.c

There are two timer functions for TCP. The *fast* timer is called every 200 ms, and the *slow* timer is called every 500 ms. The fast timer function deals with the delayed acknowledgement timer [WS95].

The fast timer checks to see if a delayed acknowledgement is waiting to be sent. It checks every 200 ms, so the delayed acknowledgement is sent between 1 and 200

ms from the time it is prepared. In addition to sending the acknowledgement, the new implementation resets the outstanding acknowledgements count (`t_acknowcnt`) to zero. It also increments the tracking variable that counts the number of delayed acknowledgements triggered by the timer (as opposed to the preset outstanding acknowledgement limit).

3. EXPERIMENTAL RESULTS

We ran three groups of experiments to test the effect of an extended acknowledgement interval on connection throughput. The first group of tests ran on the local network. The second group of tests ran on a near network, which is separated from the local network by one bridge. The third group of tests ran on a distant network over the Internet. For each test run, we transferred a large amount of data from a remote machine to the machine with the altered operating system and captured connection statistics such as throughput and number of retransmissions for subsequent analysis.

3.0.6 Statistical Analysis

In order to make a meaningful comparison between any two test brackets using confidence intervals, we need a normal distribution of data points in each bracket [Fre92]. However, we cannot assume that a set of tests measuring throughput between two given machines will yield a normal distribution. For example, figure 3.1 contains a graph of the raw data for an extended acknowledgement interval of 3.

The central limit theorem states that, given sets of random samples from an infinite population, the means of the sets are normally distributed regardless of the initial distribution [Fre92]. To compare the data from two extended acknowledgement interval settings, we must first divide tests from each acknowledgement interval setting into sets, and then take the means of the sets. The means will form a normal distribution, which can be used to compare the two settings. Figure 3.2 shows the adjusted throughput graph for the extended acknowledgement interval 3. The adjusted curve

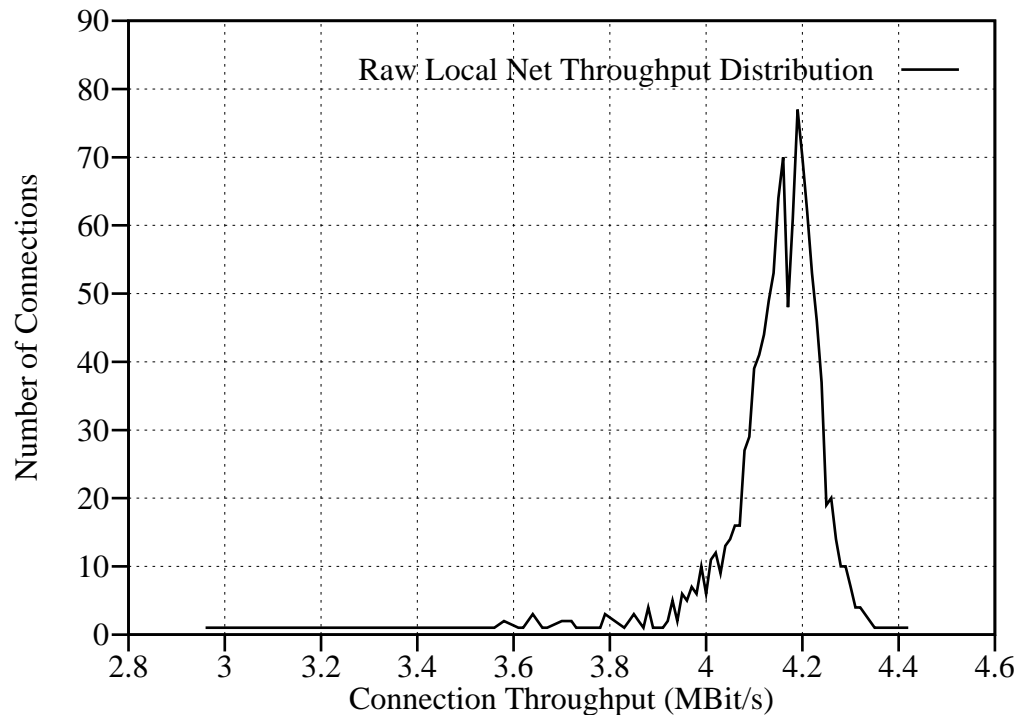


Figure 3.1 Raw Throughput Curve

This is the raw throughput distribution for the extended acknowledgement interval of 3, with a buffer size of 4k.

is approximately normal, which will allow comparisons with the adjusted throughput distribution from another acknowledgement interval setting.

Given the results of the tests, we can say with a certain confidence that the true mean of the bracket population lies within a specific range [Fre92]. For example, given the results of the local tests, we can say with a 95% confidence level that the mean of throughputs using an extended acknowledgement interval of 2 and a buffer size of 4k falls within the range of 4.062 Mbit/sec to 4.067 Mbit/sec. If the confidence intervals for two extended acknowledgement intervals overlap, then there

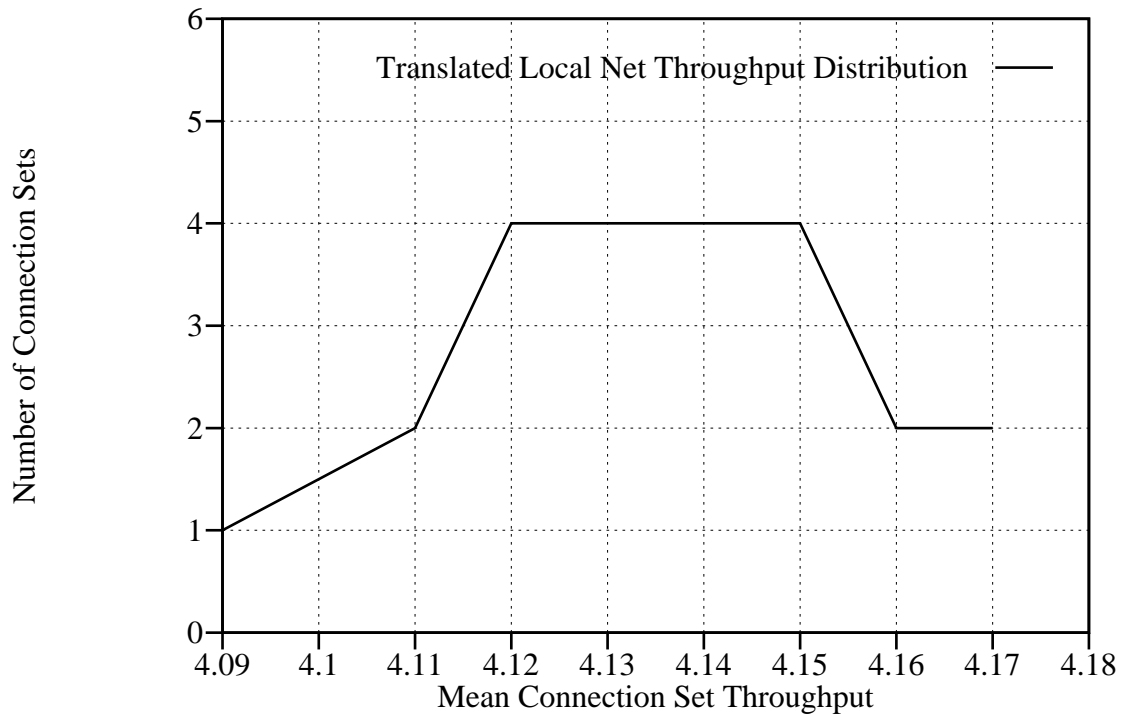


Figure 3.2 Adjusted Throughput Curve

This is the adjusted throughput distribution for the extended acknowledgement interval of 3, with a buffer size of 4k.

is no statistical difference between them, i.e. we cannot claim that one extended acknowledgement interval gets better results than the other.

3.1 Local Network

For the purposes of this research, two machines are considered to be on the same local network if they are connected to the same physical network with no bridges or routers between them. The local network tests involved three machines, which are all

connected to the same section of Ethernet. The receiving TCP was always located on netipc2, a SUN IPC workstation running SunOS 4.1.2 from Sun Microsystems, with the modifications listed in chapter 2. The sending TCP was sometimes running on netipc1, a machine identical to netipc2 but running an unmodified operating system. The other sending machine was jarok, a SUN Sparc5 running SunOS 5.3 from Sun Microsystems. Experiments ran during the early morning hours, when local network traffic is presumed low.

3.1.1 Gathering data

To measure TCP performance between two hosts, we used a program called `ttcp`. It was created at the U.S. Army Ballistics Research Lab (BRL) [Ste94]. The program produces network traffic and then presents various performance measurements, including throughput.

To measure the effect of changing the extended acknowledgement interval on the receiving machine, we ran several tests at different extended acknowledgement interval settings. We compared the results of the tests using the statistical methods outlined in section 3.0.6.

3.1.2 Experiments with a 4k Window Size

As mentioned in section 1.3, the buffer size at both ends of a TCP connection can effect throughput considerably. The size of the smallest buffer is the maximum amount of data that may be in transit at a given time. We ran the first set of experiments at TCP's default buffer size setting of 4k (4096) bytes.

Over 24 thousand connections ran between netipc1 and netipc2. The tests were gathered into sets of 20 for analysis. The graph in figure 3.3 shows the 95% confidence

intervals calculated over the normally distributed mean of each extended acknowledgement interval. Non-overlapping confidence intervals indicate a statistically significant difference in throughput. The graph indicates that acknowledging every third segment does improve throughput. However, the throughput gains do not continue to rise for extended acknowledgement intervals of four and above. There is a statistically significant dip in the throughput at extended acknowledgement interval 14. This suggests some adverse interaction within TCP at this acknowledgement interval.

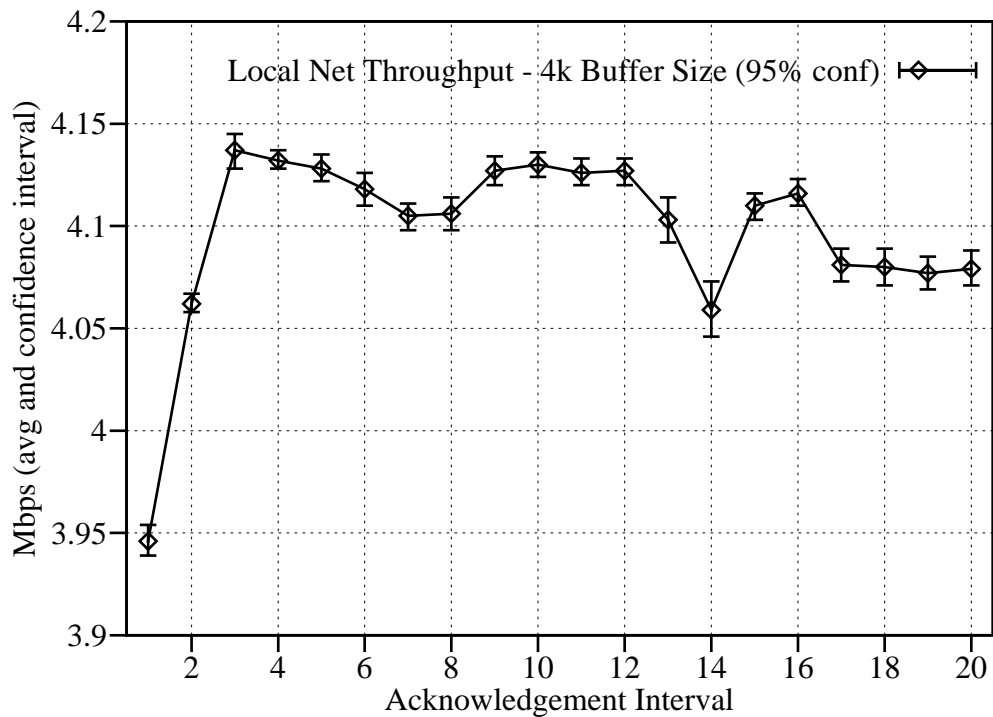


Figure 3.3 Local Network Throughput - 4k Buffer Size

Each data point represents the mean of one extended acknowledgement interval bracket. Non-overlapping confidence intervals indicate a statistically significant difference in throughput.

3.1.2.1 Explanation of Results

In the local network case, extended acknowledgement intervals increased throughput. Each data point on Figure 3.3 represents one extended acknowledgement interval setting. The diamond represents the throughput mean at that interval, and the vertical bar through the mean extends over the confidence interval range. We can state with 95% confidence that an extended acknowledgement interval of three increases throughput (from the default of 2) on a local network basis.

In the local network case, the RTT drawbacks of using an extended acknowledgement interval do not make a significant impact. One of the principal drawbacks of using an extended acknowledgement interval is that the RTT for the connection will be inflated. This inflated RTT will not matter unless a segment is lost or corrupted. Then, the connection will take longer to find out about the segment loss. On the local net, lost segments (and therefore retransmissions) are rare. Retransmitted segments may be detected as bytes arriving out of order in the connection; throughout over twenty-six thousand local connections, .001 percent of the data bytes appeared out of order. Thus, the inflated RTT seems to have a minimal effect on local net connections.

Having fewer packets for a given connection may allow higher throughput [Cla82]. As previously stated, one of the major sources of TCP overhead is in the interrupt time taken to deal with each incoming packet. The extended acknowledgement interval allows for fewer acknowledgement packets per connection, which cuts down on the overhead of processing packets [Cla82]. In addition, more of the available network resources can be used for data segments. Figure 3.4 shows the average number of acknowledgement segments sent from the receiving side of the test connections at

each acknowledgement interval tested. The amount of data to be transferred and the window size were constant throughout the experimental runs.

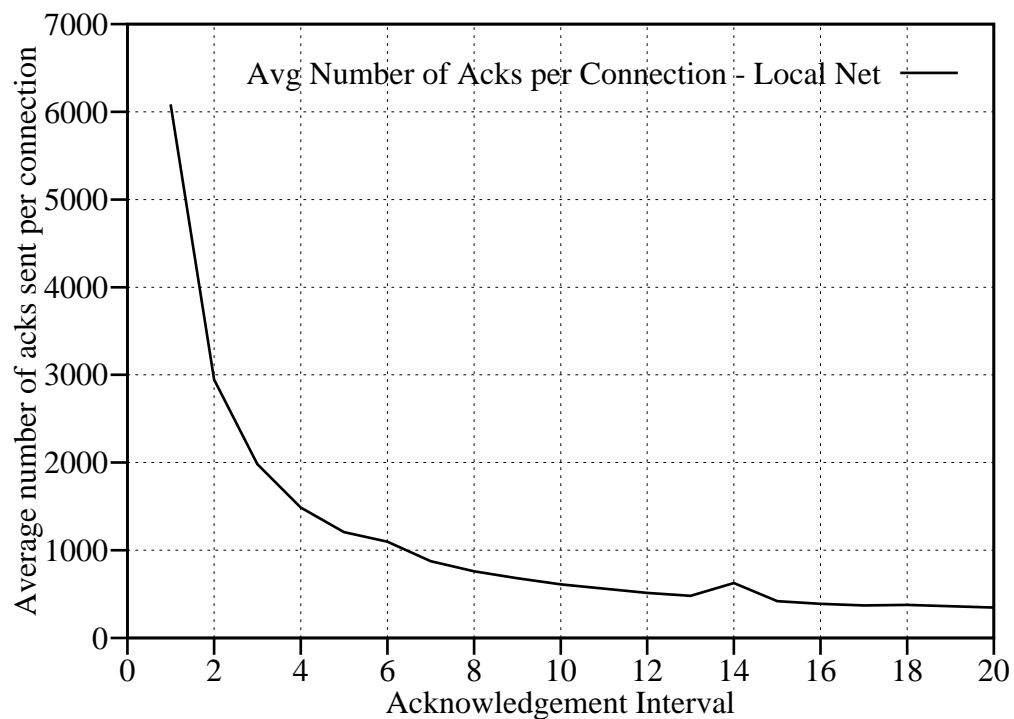


Figure 3.4 Average Number of Acknowledgements per Connection

The average number of acknowledgements per connection drops as the extended acknowledgement interval increases. Fewer acknowledgement segments per connection means less interrupt overhead for processing the incoming packets of a connection.

Throughput levels off after an extended acknowledgement interval of three. Recall that the window size (sending/receiving buffer size) of the test connections is 4k bytes. After the sending TCP sends enough segments to fill the window, it will pause and

3.1.3 Experiments with a 51k Window Size

Because the 4k buffer size appeared to be limiting the usefulness of extended acknowledgement intervals, we increased the buffer size from 4k to 51k, the maximum buffer size allowed by the operating system [PP93]. Approximately 3 thousand connections were run from jarok to netipc2, and gathered into sets of 20 for analysis. We used jarok instead of netipc1 as the sending machine because we were using netipc1 to capture traces of the resulting network activity. Although jarok is a faster machine than netipc1, there are no experimental side effects from changing machines because we never compare data sets run from jarok with those run from netipc1.

The graph in figure 3.6 shows the results of the test sets running at the increased buffer size. Although the overall throughput rises for each acknowledgement interval, the extended acknowledgement intervals do not show as much improvement over the default behavior as they did at the 4k window size setting.

3.1.3.1 Explanation of Results

The mean throughput rise is a direct result of the increase in the window size from 4k to 51k (see section 1.3). The gradual increase in the throughput as the extended acknowledgement interval rises is to be expected because the window size is now large enough for the extended acknowledgement intervals to be used more frequently; the sending TCP will not have to stop and wait every few segments, as it did with the 4k window size.

The extended acknowledgment intervals are not having as significant an effect as expected on connection throughput. To discover the cause of the anomaly, we ran 10 connections at differing acknowledgment intervals and buffer sizes to get a rough idea of what was taking place. The result of this experiment is shown in figure 3.7. Although it is a simple test, it demonstrates that something is hindering the

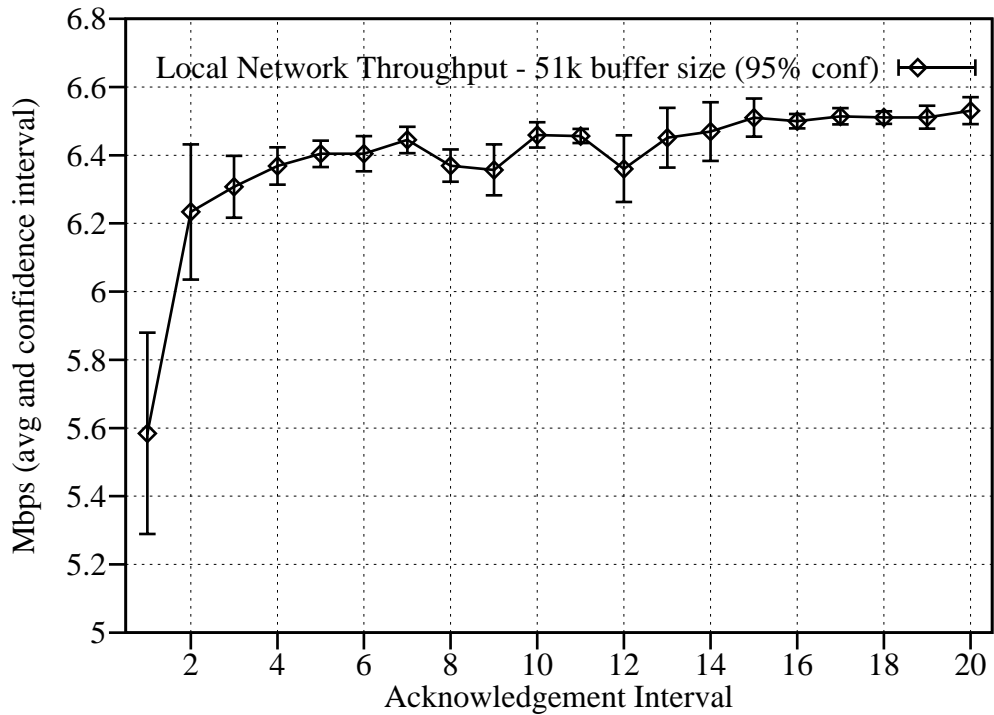


Figure 3.6 Local Network Throughput - 51k Buffer Size

Each data point represents the mean of one extended acknowledgement interval bracket. Non-overlapping confidence intervals indicate a statistically significant difference in throughput.

throughput at the maximum buffer size. The same effect has been noted by others [PP93], and is thought to be a problem with the SunOS 4.1 operating system. Based on this knowledge, we decided to run the tests with a window size of 32k to avoid the adverse effects of the maximum buffer size.

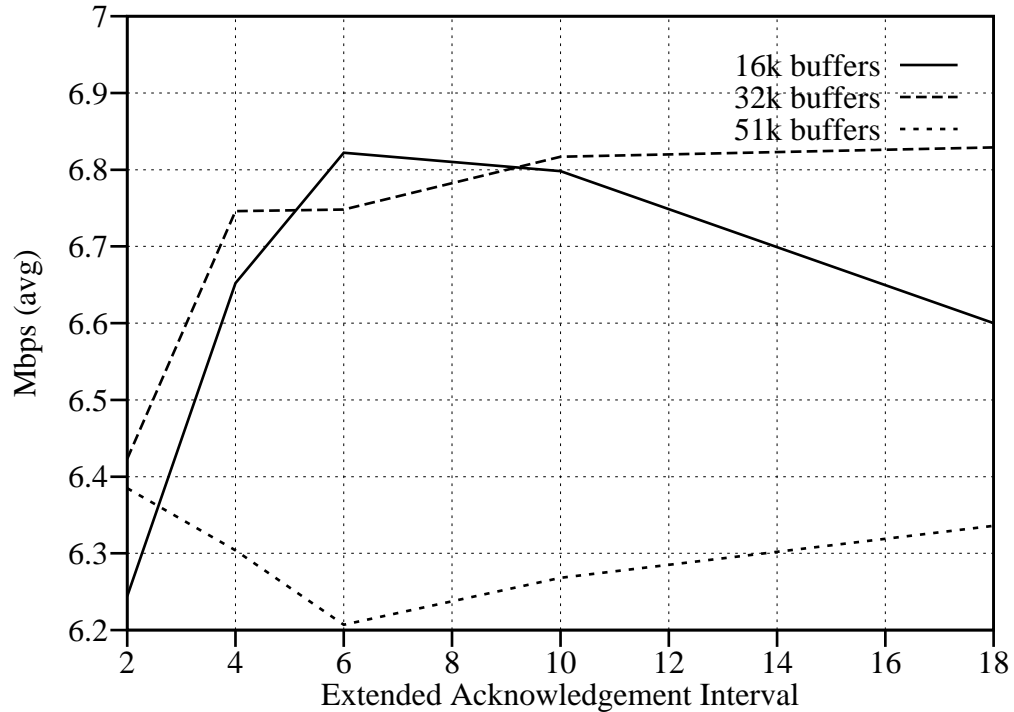


Figure 3.7 Effect of Buffer Size on Local Network Througput

10 tests were run at extended acknowledgement intervals of 2, 4, 6, 10, and 16. The means of the tests are shown in this plot. The maximum buffer size of 51k has an adverse affect on throughput. This has been noted before [PP93] and is thought to be a result of using the SunOS 4.1 operating system.

3.1.4 Experiments with a 32k Window Size

A 32k window size is large enough for TCP to send several segments before filling the window, which will alleviate the “stop and wait” problems incurred at the default window size of 4k. The 32k window size also seems to avoid the adverse effects on throughput noted with large buffer sizes in the SunOS 4.1 operating system [PP93].

We ran 2 thousand test connections between jarok and netipc2, and gathered the results into sets of 20 for analysis. The results are shown in figure 3.8.

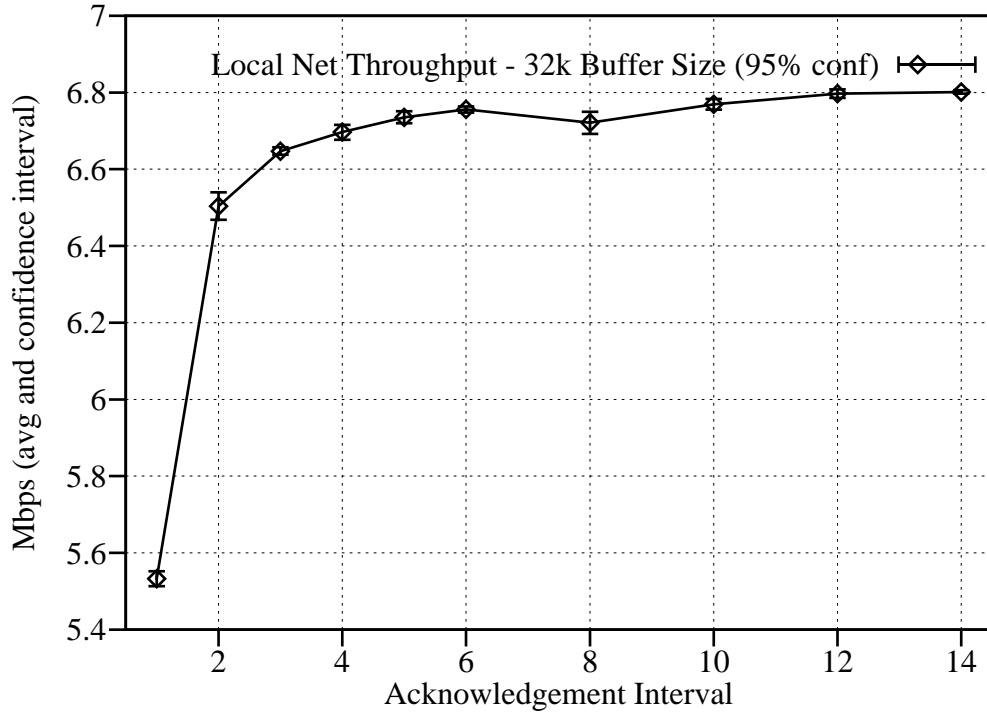


Figure 3.8 Local Network Throughput - 32k Buffer Size

Each data point represents the mean of one extended acknowledgement interval bracket. Non-overlapping confidence intervals indicate a statistically significant difference in throughput.

The extended acknowledgement intervals make a statistically significant improvement in throughput. Extending the acknowledgement interval from 2 to 3 increases the throughput by 2 percent (roughly 100 kbits/sec). Extending the acknowledgement interval from 2 to 6 increases throughput by 3 percent. We measured a 4 percent increase at acknowledgement interval 14; this was the maximum throughput gain noted in our experiments.

3.1.4.1 Explanation of Results

As mentioned in section 3.1.2.1, extended acknowledgement intervals allow for fewer acknowledgements per connection, which reduces CPU overhead in dealing with the connection. In addition, the penalties of having an abnormally long RTT measurement are rarely incurred on a local network because retransmissions are rare.

The throughput of the 32k window size connections continues to rise as the extended acknowledgement interval is raised. This did not happen in the 4k buffer space due to the “stop and wait” behavior of TCP at such a small buffer size. Figure 3.9 shows a connection running at an extended acknowledgement interval of 6. Comparing this figure to figure 3.5, one can see that the increased window size gives the extended acknowledgement interval room to work. Figure 3.10 by contrast, shows the default acknowledgement interval of 2 at a 32k buffer size. Note that several more acknowledgements are sent on the connection, and that the acknowledgements are often bunched together.

3.2 Near Network

This set of tests ran between a SparcStation 1+ running SunOS 4.1.2 from Sun Microsystems (levant) and the Sun IPC machine running the modified kernel (netipc2). The machines are on separate subnets and are separated by a bridge. The average round trip time between them is around 2.5 milliseconds.

The tests were run between midnight and seven in the morning over a period of nine days. A total of 2625 ttcp connections ran between the machines for the extended acknowledgement intervals between 1 and 14 (between 250 and 275 runs per interval). The tests were gathered into sets of 20 for analysis. Levant is on a subnet used by many people. Consequently, we ran fewer tests than in the local net

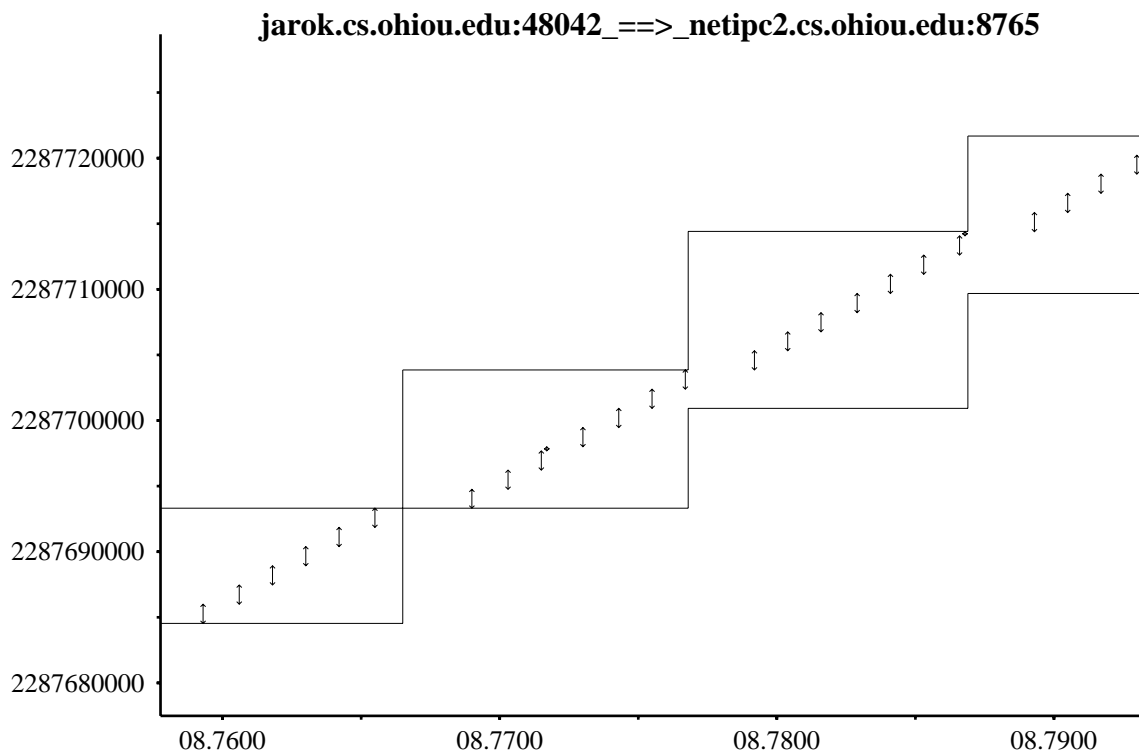


Figure 3.9 Extended Acknowledgement Interval of 6 - 32k Window Size

The use of a 32k window size allows the extended acknowledgement intervals room to work. Note that this is the same acknowledgement interval setting as shown in figure 3.5.

case. The same statistical methods apply to this data as to the local net data (see section 3.0.6).

Figure 3.11 shows the final results of the experiments between levant and netipc2. Overlapping confidence intervals indicate that there is no statistical difference in the throughput means of the default behavior and the extended acknowledgement intervals of 3, 4, and 5. However, throughput increases as the extended acknowledgement interval is increased. The extended acknowledgement interval of 6 does show statistically significant improvement over the default TCP behavior.

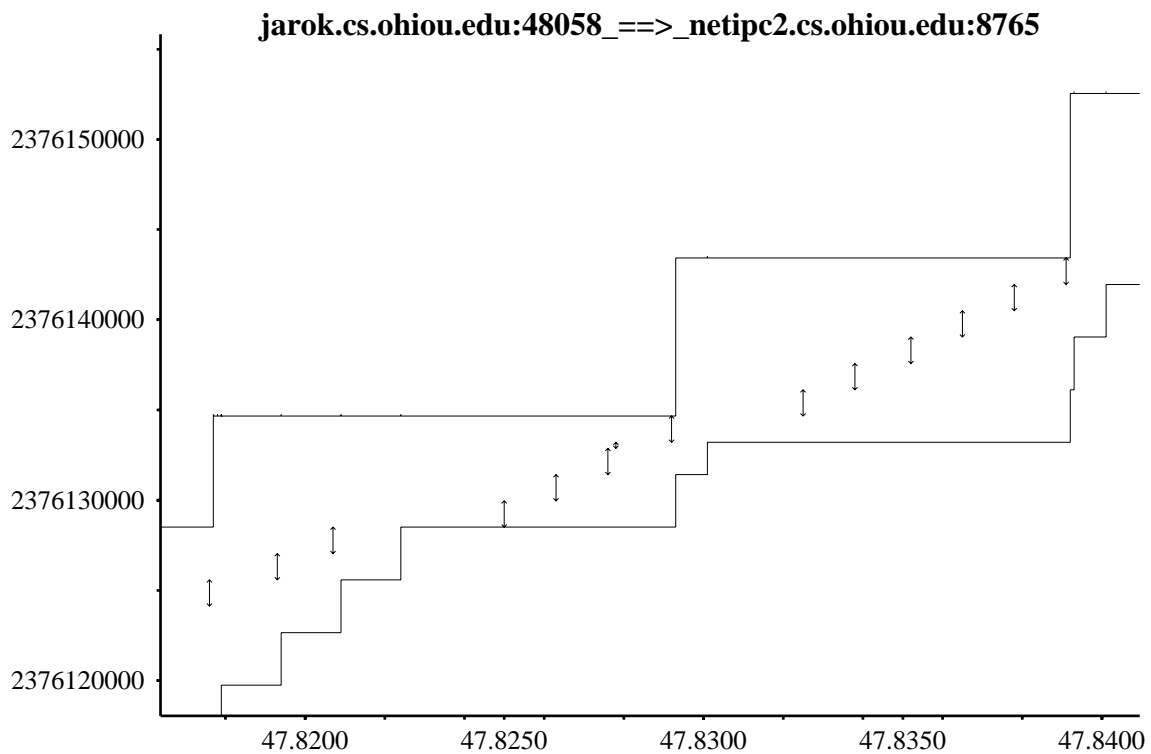


Figure 3.10 Acknowledgement Interval of 2 (Default) - 32k Window Size
 Although the receiving TCP attempts to acknowledge every other segment, the acknowledgements are often bunched together as seen at time index 47.84 on this plot.

3.2.1 Explanation of Results

Extending the acknowledgement interval on the connection improves throughput. The improvements are not as marked as in the local case. One reason for this may be the increase in retransmitted bytes throughout the connection. Over the course of the experiments .003 percent of the bytes sent were retransmitted, as opposed to .001 percent on the local network.

A wider variation in background traffic on the network may also have contributed to the changed results. The local network is dedicated to testing. However, the

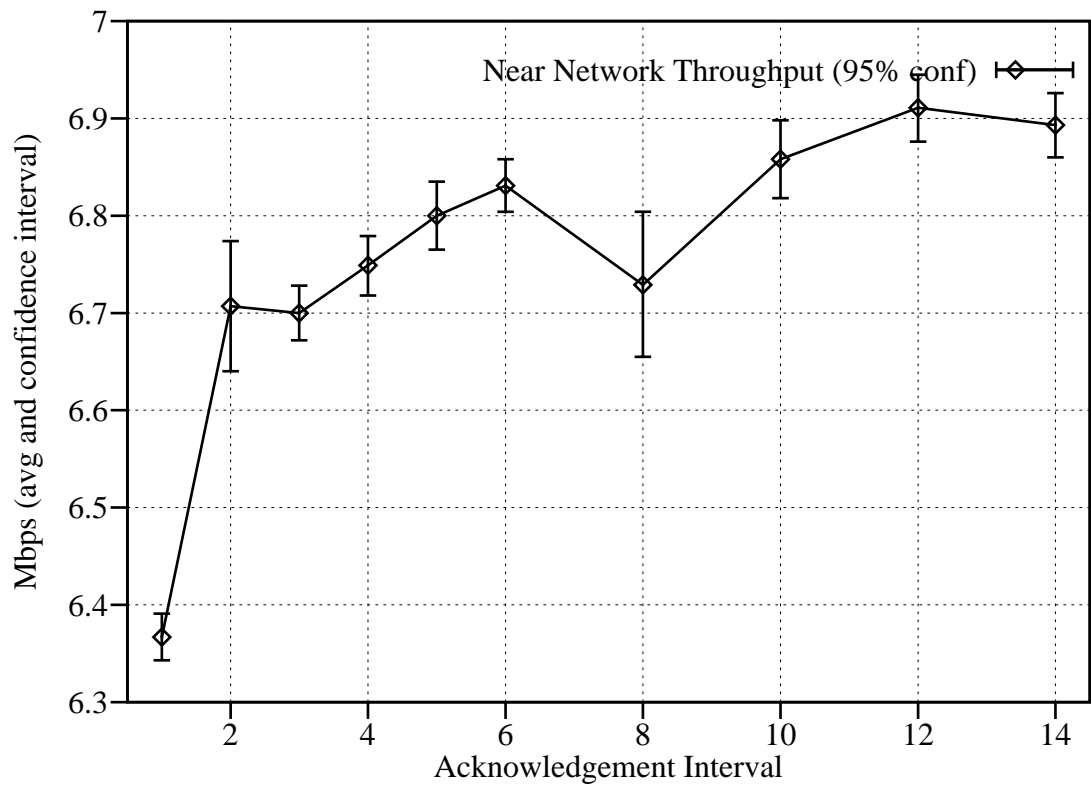


Figure 3.11 Near Network Throughput

This graph shows the throughput attained at each acknowledgement interval. The extended acknowledgement interval of 6 shows statistically significant improvement in throughput over the default behavior.

network that levant is connected is the Computer Science Department network; several users may be on at any given time. In addition, large batch processes run at night, including the downloading of news to the department server. The wider variation in background network load may have widened the confidence intervals for each acknowledgement interval setting.

3.3 Distant Network

This set of tests ran between a Silicon Graphics Indigo running IRIX 5.1.0.1 (ogion)¹ and a Sun IPC machine running the altered kernel (netipc2). Ogion is 8 gateways away from netipc2, which is about a 20-25 millisecond round trip time.

The File Transfer Protocol (FTP) was used to transfer a 1 meg² file 121 times from ogion to netipc2, approximately 30 times per acknowledgement interval. We used FTP instead of ttcp because ttcp tends to saturate the network when it is running. Tests ran for extended acknowledgement interval settings of 2, 4, 8, and 15. The same statistical methods apply to this data as to the local net data (see section 3.0.6).

The results of the experiments between ogion and netipc2 are shown in figure 3.12. The overlapping confidence intervals indicate that there is no statistical difference in the throughput means.

¹We extend our thanks to Dr. Steve Chapin at Kent State for the kind use of his machine

²1 meg = 1024k = 1,048,576 bytes

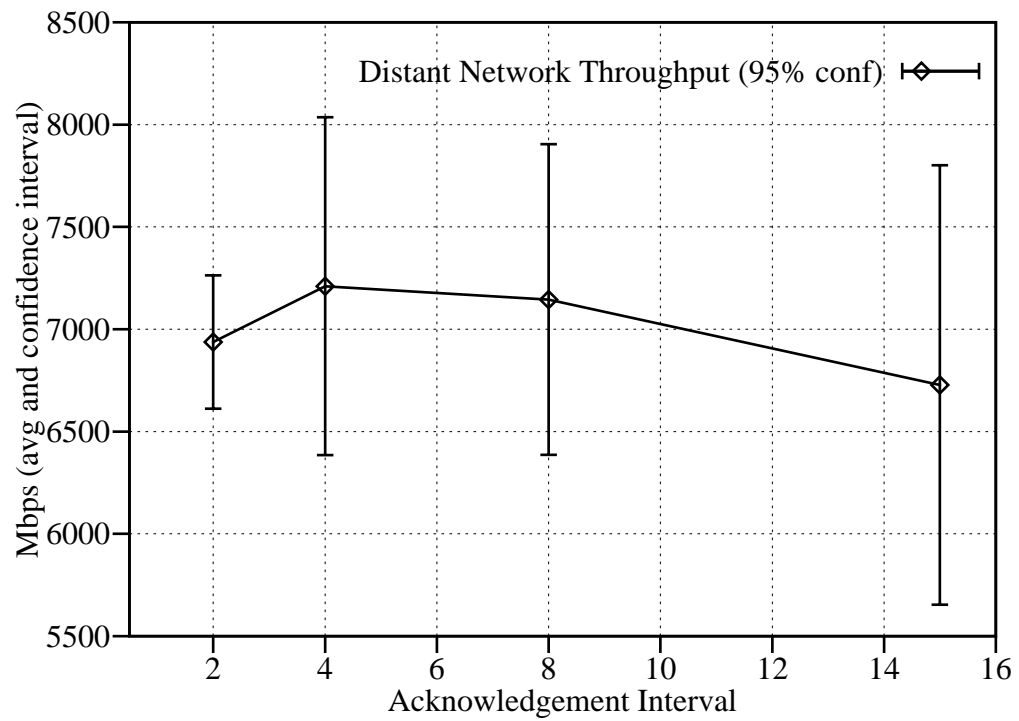


Figure 3.12 Distant Network Throughput

This graph shows the throughput attained at each tested acknowledgement interval. The hosts were 8 gateways apart over the Internet. The overlapping confidence intervals mean that there is no statistical difference in the throughput means.

3.3.1 Explanation of Results

Wide variations in the throughput for each extended acknowledgement interval setting are causing the wide confidence intervals. These variations could be caused by changing congestion conditions on the Internet.

During the experiment, about 1 percent of the segments were retransmitted or arrived out of order, compared with .001 percent in the local network case. The abnormally long RTT may be adversely affecting the connection when a segment is lost in transit.

The average number of segments per connection declines over the extended acknowledgement interval range from 2 to 8. At the extended acknowledgement interval of 15 the average number of segments rises, indicating a higher percentage of retransmissions.

4. CONCLUSIONS

4.1 Overview

TCP is a reliable network protocol that uses the unreliable IP protocol to deliver data. TCP uses positive acknowledgements in conjunction with retransmissions to provide reliability. Because the acknowledgements are cumulative, some of them may also be redundant. We hypothesize that using an extended acknowledgement interval may eliminate some redundant acknowledgements, thereby increasing the throughput of the TCP connection.

We altered a UNIX operating system on one machine to use extended acknowledgement intervals; instead of acknowledging every other TCP segment, it acknowledges every n segments. We ran throughput tests over a local network, a near network, and a distant network. In the local and near network cases, we captured statistics on thousands of bulk data connections made using `ttcp`. In the distant network case, we tested the throughput using `ftp` connections due to `ttcp`'s tendency to saturate the network.

4.2 Conclusions

Extending the TCP acknowledgement interval increases throughput in some cases. In particular, extending the acknowledgement interval increases throughput when

two machines are on relatively close networks. Using extended acknowledgement intervals artificially inflates the RTT for each segment sent along the connection. Consequently, the sending TCP may take longer to realize when a segment has been lost or damaged. This effect may counteract any throughput gains caused by the extended acknowledgement intervals. This would help to explain why the extended acknowledgement interval throughput gains observed in the local and near network cases were absent in the distant network case¹.

The majority of the throughput gain may result from the fact that fewer packets are needed for data transfers when using an extended acknowledgement interval. This cuts down on the amount of interrupt overhead in dealing with the connection. In addition, the use of fewer packets for a given connection is considered “network friendly”.

Most of the throughput gain will be found in the lower range of extended acknowledgement intervals (between 3 and 10). As the extended acknowledgement interval setting is increased, the increase in throughput from one interval to the next will diminish. The number of time-triggered acknowledgements will increase as the extended acknowledgement interval rises.

4.3 Recommendations for Further Research

The research presented here is based on a static extended acknowledgement interval. If the extended acknowledgement interval is relatively high (4 or more), then the connection will tend to have poor throughput when the network becomes congested. It may be possible to make the extended acknowledgement interval a dynamic variable that changes throughout the connection lifetime to adapt to network congestion.

¹The amount of retransmissions in the distant network case were approximately 100 times the local network case.

Research looking into a dynamic extended acknowledgement interval will have to consider what to do when the network becomes congested. One approach would be to back off to a lower acknowledgement interval when the network becomes congested. This would allow closer monitoring of the connection until the congestion passes². This process would mirror the TCP congestion window algorithm. The research which has been done on this algorithm may directly apply to a dynamic extended acknowledgement interval. The overhead of implementing such a “slow-start” scheme must be weighed against the possible throughput gains made possible by having fewer packets per connection.

One possible problem with implementing dynamic intervals will be its effect on the RTT. A stable connection running at a high extended acknowledgement interval will have inflated RTTs. When the interval is lowered, it will take several RTT samples for the RTO to catch up with the drastic changes in the RTTs. Conversely, when the interval is increased, the lack of immediate change in the RTO may cause unnecessary retransmissions as the RTT suddenly doubles or triples.

Another option for using the extended acknowledgement intervals is to modify the receiving TCP to recognize whether a connection is local, and to use the extended intervals on the local connections. Much of daily bulk transfers is between local machines; using extended acknowledgement intervals locally will improve these transfers.

The use of extended acknowledgement intervals to increase TCP throughput shows promise. We have shown significant throughput increases in local and nearby networks. With some alterations, it may be possible to increase throughput in the distant network case.

²Several connections backing off to a lower acknowledgement interval will increase acknowledgement traffic, but it will be in the opposite direction from the congestion. This may still be a problem if the network is congested in both directions.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [BOP94] Lawrence Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM*, pages 24–35. SIGCOMM, October 1994.
- [Bra89] R. Braden. Requirements for Internet Hosts - Communication Layers, October 1989. RFC 1122.
- [Cla82] D. D. Clark. Window and Acknowledgement Strategy in TCP, July 1982. RFC 813.
- [Com88] Douglas E. Comer. *Internetworking with TCP/IP Volume I, Principles, Protocols, and Architecture*. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
- [Fre92] John E. Freund. *Mathematical Statistics*. Prentice-Hall, Englewood Cliffs, New Jersey, 1992.
- [Jac88] V. Jacobson. Congestion Avoidance and Control. In *Proceedings ACM SIGCOMM '88*, pages 314–329. ACM, August 1988.
- [JBB92] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance, May 1992. RFC 1323.
- [LMKQ89] S. Leffler, M. Mckusick, M. Karels, and J. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, Reading, Massachusetts, 1989.
- [Pos81] J. Postel. Transmission Control Protocol, September 1981. RFC 793.
- [PP93] Christos Papadopoulos and Gurudatta M. Parulkar. Experimental Evaluation of SUNOS IPC and TCP/IP Protocol Implementation. *IEEE/ACM Transactions on Networking*, 1(2):199–216, April 1993.

- [Ste94] W. Richard Stevens. *TCP/IP Illustrated, Volume 1, The Protocols*. Addison-Wesley, Reading, Massachusetts, 1994.
- [WS95] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume 2, The Implementation*. Addison-Wesley, Reading, Massachusetts, 1995.

APPENDIX

A. HOW TO READ A TCP TIME SEQUENCE GRAPH

A TCP time sequence graph is a graphic representation of what is happening during a TCP connection. Recall that TCP connections are full-duplex; data can flow in both directions simultaneously. Each time sequence graph shows only one side of the connection. This is ideal for our situation; our experiments only send data in one direction. Figure A is an example of a TCP time sequence graph.

Each side of a TCP connection consists of data flowing from the sending TCP to the receiving TCP and acknowledgements flowing from the receiving TCP to the sending TCP. Recall that each data segment sent along the connection has a TCP sequence number. The y-axis of the TCP time sequence graph is the TCP sequence number. The x-axis of the graph is the time index of the connection in seconds. Any graphed connection will have a diagonal movement along the page from lower left to upper right; sequence numbers increase as time goes on.

Each segment sent along the connection is represented by a vertical line with arrows at each end. The height of the line is the size of the segment in bytes. The vertical placement of the line on the page corresponds to the segment numbers of the bytes that are covered in that segment.

The horizontal line over the segments represents the top of the receiving window. The total size of the receiving window is the space between the horizontal lines containing the segments. Graphically, the segments must exist within the receiving TCP's window.

Examine the top edge of the receiving window, which is the top horizontal line. Each time the top of the receiving window advances along the sequence space, there

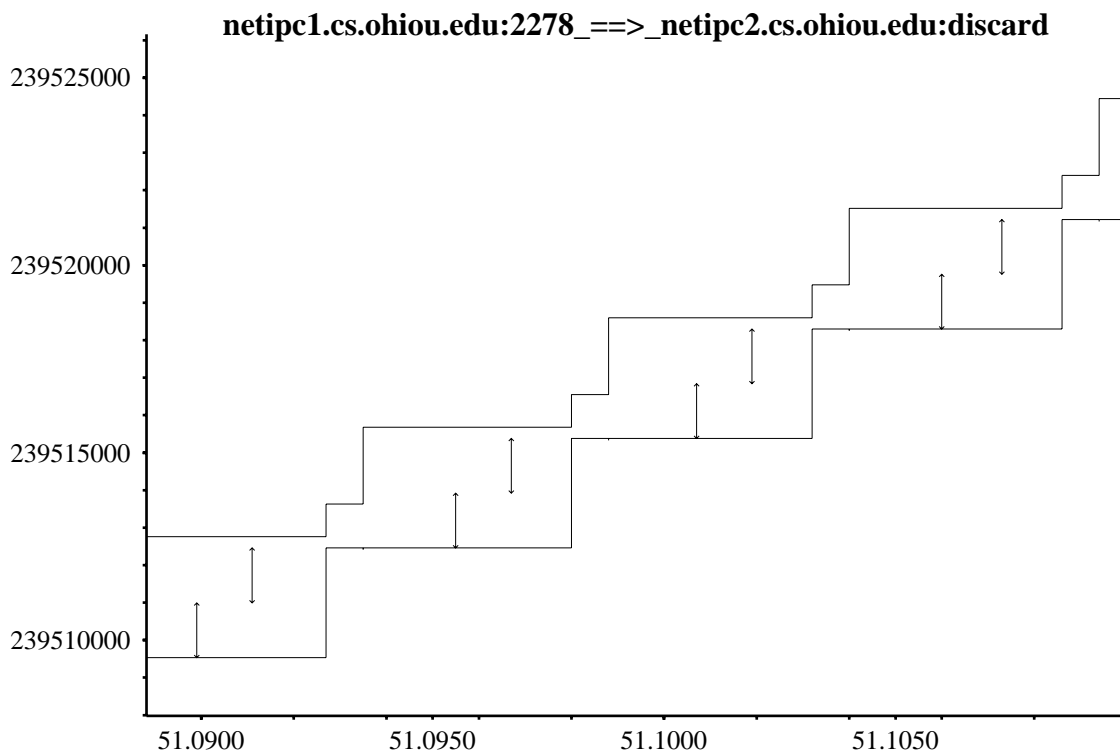


Figure A.1 Example TCP Time Sequence Graph

This graph represents one side of a TCP connection. The x-axis measures the sequence numbers, and the y-axis measures time in the connection (seconds).

is a vertical line. This line represents an advance of the receiving window along the sequence number space - i.e. a TCP window update. In general, this means that the application on the receiving side read data from the receiving TCP buffer.

Examine the bottom edge of the receiving window, which is the bottom horizontal line. Each time the bottom of the receiving window advances along the sequence space, there is a vertical line. This line represents an acknowledgement which covers the segments up to and including its position on the sequence space. For example, in figure A the acknowledgement shown at approximately 51.098 seconds covers the third and fourth segments shown in this graph.