Measuring TCP Congestion Control Behaviour in the Internet

Stephen Eichler

This report is submitted in partial fulfilment of the requirements for the degree of Master of Science at the University of Waikato.

February 28, 2011

Abstract

The Internet is constantly changing and evolving. In this thesis the behaviour of various aspects of the implementation of TCP underlying the Internet are measured. These include measures of Initial Congestion Window (ICW), type of reaction to loss, Selective Acknowledgment (SACK) support, Explicit Congestion Notification (ECN) support. We develop a new method to measure congestion window reduction due to three duplicate ACK inferred loss. In a previous study 94% of classified servers showed window halving, whereas we found that 50% of classified servers exhibited Binary Increase Congestion control (BIC) or Cubic style behaviour, which is a departure from a Request For Comments (RFC) requirement to reduce the congestion window by at least 50%. ECN is predicted to improve Internet performance, but previous studies have revealed a low support for it 0.5%, and ECN connections failed at a high rate due to middlebox interference 9%; in this thesis we show a steady increase over time of ECN being implemented and supported 7.2%-10.3%. ECN testing of webservers with globally routable IPv6 adderesses showed a higher success rate 21.9%. Analysis of congestion control behaviour such as Tahoe, Reno and New Reno showed New Reno dominating more strongly than before, increasing from 35% to 70% of popular webservers.

SACK sending analysis revealed that 45% of popular webservers implement it properly, as compared to 18% in earlier studies. SACK receiving analysis showed higher results to the earlier studies, with success increasing from 64.7% to 81.1%. For both of these SACK studies results for webservers with globally routable IPv6 addresses showed a higher success rate when errors remained low. Analysis of ICW indicates that 75% of popular webservers implement the older ICW regime of an initial congestion window of two or less packets, as compared to 96% in previous studies. The new regime of an ICW of three or four packets depending on segment size was implemented at 20%. We see from these results that RFCs do affect TCP implementation, but change can be slow. However we see that implementation and support for modern TCP features is increasing.

Acknowledgements

I would like to thank the following people and organisations for their assistance with this thesis:

Dr Matthew Luckie from the WAND Group - my supervisor.

The Waikato University scholarship office - for funding me through a Masters Research Scholarship.

Shane Alcock and Brendon Jones from the WAND group - for assistance with programming and assembling the controlled environment.

Contents

List of Acronyms

AS Autonomous System

ABC Appropriate Byte Counting

BIC Binary Increase Congestion control

CE Congestion Experienced

CWR Congestion Window Reduced

ECE ECN Echo

ECN Explicit Congestion Notification

ECT ECN Capable Transport

IP Internet Protocol

ICW Initial Congestion Window

MSS Maximum Segment Size

MTU Maximum Transmission Unit

PMTUD Path MTU Discovery

RED Random Early Detection

RFC Request For Comments

RTO Retransmission Time Out

RTT Round Trip Time

SACK Selective Acknowledgment

TBIT TCP Behaviour Inference Tool

TCP Transmission Control Protocol

WAND Waikato Applied Network Dynamics

Chapter 1

Introduction

1.1 The Problem

Transmission Control Protocol (TCP) is a critical Internet protocol yet there is a lack of knowledge about how the TCP behaves "in the wild". This may be because it has largely been accepted by users as a black box [50] [47], that works at an acceptable level most of the time. In order to maintain stability and to improve performance it is important to know how it behaves. Do TCP implementations behave in the way the Request For Comments (RFC) documents say they should? This is important because these RFCs are intended to specify changes that improve Internet behaviour [4, 20, 34]. In particular there is a lack of recent information about the state of TCP in the Internet [36].

TCP is a complicated transport protocol because its features and behaviour have evolved over time [4, 53, 34, 43, 45, 1, 21, 2]. There is always a delay between the design of a new TCP algorithm and its appearance in operating systems. Accepted changes are slower if they involve changes to network hardware such as ECN [36], rather than being strictly "end to end"

involving only end host operating system software. It is of interest to detect and understand different rates of TCP modification uptake, because it may prove useful in improving prediction of future uptake of new TCP features. Furthermore it has proven difficult to predict these time frames of change with any degree of accuracy [10].

There is a lack of tools to analyse the TCP behaviour of the entire Internet. However there are tests that can analyse many aspects of subsets of Internet TCP. There has been fairly limited analysis of TCP change on the Internet, and active probing suites of TCP evaluations have largely not been reported since 2001 and 2004 [36]. Furthermore, there has not been comprehensive extension of these tests to cover newer TCP characteristics. This means that there are some useful analyses that have not yet been put into practise. For example, varying negotiated Maximum Segment Size (MSS) when testing Initial Congestion Window (ICW), or analysing packet flights when analysing congestion window reduction due to loss inferred by three duplicate acknowledgements.

1.2 Specific questions

The behaviours of TCP that were chosen to study are related to the ability of TCP to cope with congestion and hence the performance of the Internet under heavy load.

When loss is experienced as inferred by three duplicate acknowledgements, the congestion window should reduce by half as specified in RFC 2581 [4]. An important behaviour of TCP in preventing congestion collapse

is that this reduction is to lessen the traffic at points in the network where packet loss is occurring, the initial stage of congestion collapse [38], and prevent congestion collapse from occurring. Congestion collapse is a state where there is high loading demand and very little network throughput. So does this congestion window reduction occur, or do smaller reductions occur in some cases? It is desirable to know if modern TCPs conform to this expectation, and in addition which operating systems do and do not behave in the expected way as congestion collapse results in severe loss of Internet performance [38].

Another possibility for controlling congestion is through using Explicit Congestion Notification (ECN) [45]. This method controls congestion without dropping packets by having routers mark packets before congestion occurs. This is an improvement on Random Early Detection (RED) which drops packets with a probability associated with queue size. In response to the ECN marked packets the notified sender should then behave as though a packet were dropped reducing its congestion window. The question of quantifying ECN uptake and its accurate implementation is important because ECN is intended to improve TCP performance by reducing packet loss but previous studies [36] have revealed impediments, such as high rate of failure of ECN capable connections which was largely atributed to middleboxes.

The ICW setting also affects the control of Internet congestion. It is important that senders start at a small congestion window value to prevent the sender from causing congestion when it starts transmitting [4]. RFC 5681 [2] specifies an ICW of 2 packets for MSS greater than 2190 bytes, ICW of 4 for MSS less than or equal to 1095 and otherwise ICW is 3. In simple

terms the initial window is limited to 4380 bytes [1]. Some advantages of increasing ICW are that receivers using delayed ACKs do not have to wait for a time out, senders transmitting a small amount of data may complete their transmission very quickly, and senders able to open to large congestion windows may save as much as three RTTs and the delay mentioned in the first case [1]. So, what proportions of ICW values do we observe in the Internet, and do they conform to guidelines?

Selective Acknowledgment (SACK) is a means to prevent loss of throughput when multiple packet losses occur. The receiver sends information about which packets have been received successfully to the sender. The sender responds by sending the specific packets which are missing. In the past partial SACK implementation has been observed and cases of incorrect SACK implementation [36]. As these factors affect TCP performance, it is useful to ask what changes in SACK implementation have occurred on the Internet, in order that operating system designers might have more information about how the SACK algorithm is performing on existing systems.

Finally, there are a number of flavours of TCP loss recovery which occur in the absence of SACK. TCP loss recovery is part of the congestion control mechanism, however an important difference between these flavours of TCP is the effect on throughput. Tahoe responds to loss in the same way as a timeout, Reno is designed to deal with single packet drops, and New Reno deals better with multiple drops. As New Reno has better performance than Reno, and Reno in turn than Tahoe, what is their relative prevalence?

1.3 Overview of Thesis

The problem addressed by this research is outlined in this chapter, and in the subsequent chapter relevant background to this work is presented.

Chapter 3 examines congestion window behaviour under loss as inferred by multiple duplicate acknowledgements; we find a large increase in BIC and Cubic behaviour and a corresponding reduction in regular TCP behaviour [36]. Chapter 4 examines the implementation of ECN in the Internet; we find a steady increase in ECN server support over several months (Sep 2010 - Feb 2011). Similarly chapter 5 reports on analysis of Tahoe, Reno and New Reno flavours of TCP, we find a large increase in New Reno conformant webservers. Chapter 6 analyses for adoption of SACK; we find a large increase in correct implementation of SACK as determined by the sender SACK test (webserver SACK usage) relative to the level measured in 2004 [36]. Sender SACK is where the ability of the receiving webserver to interpret SACK blocks is determined. There is a moderate increase in correct SACK implementation as determined by the receiver SACK test (webserver SACK generation). Receiver SACK is where the ability of the webserver under test to send correct SACK blocks is determined. Chapter 7 looks at measures of initial congestion window, we find that there has been an increase from 2% [36] to 20% adoption of the new standard, of 2, 3 or 4 packets in the initial congestion window. Chapter 8 discusses conclusions and new questions which arise from these findings.

1.4 Contribution

This thesis makes several contributions to the area of studying TCP dynamics in the wild.

- A congestion window analysis algorithm has been created, which deals with non-conformance with RFC 2581 [4] and RFC 5681 [2]. When necessary focus has been shifted to a second packet drop in the algorithm which elicits more conventional TCP behaviour than the first, making the algorithm more useful for analysis. In order to carry out this analysis, profiles of congestion windows or flights have been collected and analysed across a wide variety of operating systems, though the final analysis still focuses on the congestion window reduction, after a loss event. This involves repeated measurements of congestion windows in a packet stream.

- Further data points have been added to the profile of changing Internet behaviour, as these tests have not been carried out recently. The tests include initial congestion window according to recent RFC 5681 [2], adoption of SACK, adoption of ECN and adoption of New Reno or New Reno like TCP. Analysis has also placed emphasis on detecting misbehaving implementations, and also detecting the effects of middleboxes.

- The TCP Behaviour Inference Tool (TBIT) ICW test was extended to include packets of different sizes, and an expected initial window made up of different numbers of packets.

- A number of TBIT algorithms have been modified to run as part of a modern Internet analysis tool. This involved adapting the algorithms to run in an event driven environment where several tests can be run concurrently. The structure was also modified in order to store relevant data in binary warts files including packet traces, rather than simply collecting results from standard text output from the running test program.

Chapter 2
Background

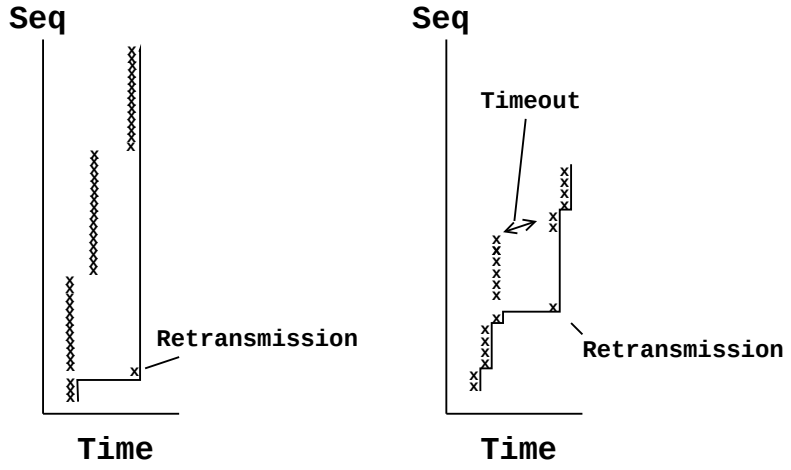
2.1 Introduction

This chapter contains a discussion of various facets of TCP important to this work and a discussion of prior work including measurements of TCP behaviour with TBIT and other methods such as measuring the behaviour of large numbers of packets in a test bed. There is also an introduction to scamper, an event driven active measurement tool, the controlled environment used to try out my scamper tests on a selection of operating systems, and the area of web server OS classification.

2.2 Overview of the Transmission Control Protocol (TCP)

TCP is the most popular transport protocol on the Internet supporting the World Wide Web, email and file transfer, and is thus a critical part of the Internet. TCP provides a reliable byte stream service from an application on

one host to an application on another host over the Internet [44]. This includes the establishment of connections, and their tear-down after use. There is bidirectional sequence numbering, reordering of out of sequence segments and deletion of duplicate segments. TCP uses acknowledgements to confirm that packets have arrived [11], and a Retransmission Time Out (RTO) timer to determine when to resend unacknowledged packets [43]. TCP retransmits a lost segment if no acknowledgment arrives, and detects corrupted segments by using checksum error detection [9]. These features of TCP originated because of the need for reliability [48]. TCP regulates its contribution to network congestion in order to minimise packet loss and reordering, and to prevent congestion collapse [2].



(a) Transmission without congestion window

(b) TCP with congestion window

Figure 2.1: Figures (a) and (b) show transmission without and with a congestion window. Both graphs show retransmission occurring. Without a congestion window transmission proceeds at a constant rate irrespective of loss, that is packet flow goes to the maximum allowed by the receive window or the maximum that the network is capable of transmitting. TCP with a congestion window in this case shows slow start behaviour.

The purpose of using congestion windows is to limit traffic or packets in flight to a level that minimises congestion events in the network. Figure 2.1 shows TCP behaviour with and without congestion windows. For each end of a TCP connection a *congestion window* [2] is kept, and it is used to limit the number of unacknowledged packets in flight from sender to receiver. A mechanism called slow start has the goal of quickly ramping up congestion window size until the point at which congestion begins to occur can be deduced. Slow start [4] involves a MSS sized increment of the congestion window for each packet acknowledged, resulting in an exponential increase of the congestion window, after connection initialisation or connection timeout. RTO is used to determine when to retransmit a packet, without waiting for a time that is either too long and wasting unnecessary time, or too short and triggering unnecessary packet retransmission. The RTO is calculated from estimates of Round Trip Time (RTT) [43]. The slow start threshold (ssthresh) is usually initialised to an 'arbitrarily high' value and then reduced to half the flight size when loss, timeout and retransmission occur. When the congestion window exceeds ssthresh, the state changes to congestion avoidance [53]. In this state there is a MSS sized increment of the congestion window for each RTT elapsed, so the congestion window grows linearly rather than exponentially.

When loss is signaled by the presence of three or more duplicate acknowledgements, the goal is to avoid unnecessarily returning to slow start, as illustrated in Figure 2.2, because packets are still being acknowledged meaning that congestion is not severe and a more moderate congestion control method may be used. When this type of loss occurs the state changes to fast retransmit then fast recovery [4]. Here the missing packet is retrans-

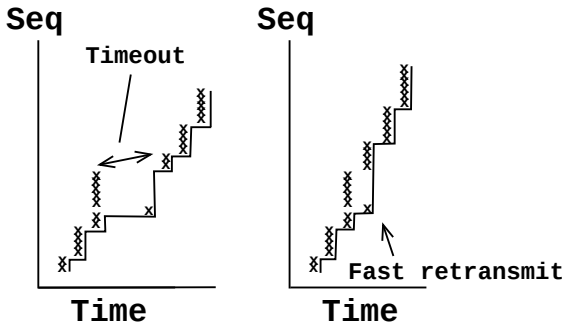
mitted before time out occurs and ssthresh is reduced, usually to half the present congestion window but at least 2*MSS [2]. In fast recovery, congestion window incrementing occurs similarly to congestion avoidance. When a cumulative acknowledgement of outstanding packets already sent is received, the state changes to congestion avoidance [26] and the congestion window is set to ssthresh. Fast retransmit and fast recovery originated after the timeout mechanism, as a result of the need to maintain TCP throughput in conjunction with balancing the need to control congestion [15].

Appropriate Byte Counting (ABC) [3] deals with the problem of the reduced numbers of ACKs associated with delayed ACKing. Delayed ACKing results in a reduction in the rate at which the congestion window opens, due to these less frequent ACKs. In ABC the congestion window is increased based on the number of bytes ACKed rather than the number of ACKs.

2.3 Overview of the Internet Protocol (IP)

Internet Protocol (IP) is a protocol that occurs in layer 3 of the ISO reference model [24]. This layer is one of four of these layers also referred to in RFC 1122 [8] and is known as the network layer. IP provides end-to-end delivery of packets from network to network, via the use of routing. This delivery protocol only makes a best effort attempt at packet transmission. Data segments from a number of upper layer protocols can be carried by IP, including TCP which provides reliability. TCP is a transport layer protocol, or an ISO layer 4 protocol.

There are two forms of IP: IPv4 and IPv6. IPv4 addresses are contained in 32 bits or 4 bytes, which allows at most 4,294,967,296 addresses. The supply of IPv4 addresses is running out and the use of the alternative IPv6 protocol [13] is growing [27]. IPv6 uses 128 bit addressing, which relates to a vast number of addresses.



(a) Without fast retramsmit

(b) With fast retramsmit

Figure 2.2: Figures (a) and (b) show TCP with and without a fast retransmit. In TCP without fast retransmit there is a timeout followed by slow start.

2.4 Measurements of TCP behaviour

2.4.1 TBIT overview

The typical situation on the Internet where active measurement occurs is where web servers act as data senders and web clients are receivers [12]. The web client is the probing or interrogating and analysing program. TBIT, an active Internet probing and analysis program, has been used to test web servers for a number of characteristics [36] [39] [40] [35]. These tests include web server SACK generation, ECN capable connections, Path MTU

Discovery (PMTUD), IP options, TCP options, Reno/New Reno discrimination, web server SACK usage, initial congestion window, congestion window reduction, byte counting, limited transmit and time wait duration. The IP and TCP options tests involve testing several options including a made up one, to see if a connection is established and to see if the option is ignored or not.

In general, a connection was established with a webserver and specific packets were sent to the webserver to elicit a response that could be analysed to determine if that system correctly implements a TCP algorithm in question. For some tests it was possible to draw conclusions about interference by middleboxes.

Reasonable levels of SACK implementation were seen, and a low but significant level of middlebox interference. This is good progress. For ECN low levels of implementation were seen, and decreasing interference by middleboxes from a moderate level. Not much progress is evident, and middlebox interference has been part of the cause of this reluctance to activate ECN. PMTUD was successfully implemented at a moderate level, and a moderate level of middlebox interference was seen. This is good progress in spite of middlebox interference.

New Reno was seen to dominate over the other types of response to loss, at a moderate level. This is good progress. Initial congestion windows were mostly one or two packets. This is a very good level of implementation.

Congestion window halving was seen to be most of classified webservers, a very good level of implementation. Appropriate byte counting was seen at a very low level, thus there has been very little progress made in implementing

this. Limited transmit was seen implemented at a moderate level. This is a good level of implementation progress.

2.4.2 Related work

There have been a number other contributions to TCP measurement besides that of the TBIT authors. In a report by Langley [30] their experiments measure responses to SYN packets with payloads where 9% of hosts do not respond, SYN packets with non standard options where 0.2% of hosts do not respond and SYN packets which attempt to negotiate ECN where 0.6% of hosts do not respond. Feyzabadi *et al.* [17] [18] analyse congestion control algorithm by looking at a series of congestion window measurements. They detect Reno, BIC and Cubic using large data sets in a controlled laboratory environment, but this approach is not well suited to being applied to the mostly smaller data sets available on Internet web servers. Fonseca *et al.* [23] found that consistent drops associated with IP options were found in a small group (12%) of Autonomous System (AS). Ladha *et al.* [29] examined five TCP enhancements by probing the top Alexa 500 webservers with TBIT: SACK, initial congestion window, limited transmit, appropriate byte counting and early retransmit. SACK negotiated was 69%, SACK information used 18%, ICW window of one packet at 128 bytes was 12%, two packets 62%, three packets 5%, four packets 3% and greater than four packets 2%, limited transmit support was 20%, appropriate byte counting support 24% and early retransmit support 0%. Synscan is an OS finger-printer that measures the type of congestion control used, among a raft of other tests. Luckie *et al.* [31] [32] analysed PMTUD failures also using active measure-

ment. This work found that failure in some cases was not caused by firewalls discarding ICMP "Packet too big" messages as previously thought. Arlitt *et al.* [5] carried out analysis on reset behaviour of TCP. This work showed that 15-25% of TCP connections contain a RST packet. Beverly [7] used a naive Bayesian classifier to also carry out OS finger-printing of packet headers. This method worked better than rule based finger-printing when data was incomplete. Bellardo *et al.* [6] used active probing extensions to sting to measure packet reordering rates. In this work reordering was measured in both directions of the TCP connection.

2.5 Scamper

Scamper is an event driven Internet active measurement tool. This means that the program operates using a list of future events with associated times, and that the programmer can specify a time when an event should occur. It also means that a number of target hosts can be analysed together. This is called a window of active probing tests, and the value for this is set from the command line. Some scamper modules require a firewall called IPFW, which is available on MacOS X and FreeBSD operating systems. This is used to prevent the scamper host operating system from responding to packets received during the running of a test, interfering in the measurement.

In order to abstract implementation from the test module, I made changes to the infrastructure of scamper to allow test modules to carry out fundamental activities in support of this thesis. For example the sending and receiving of the SACK permitted option in SYN and SYN ACK packets. Modifica-

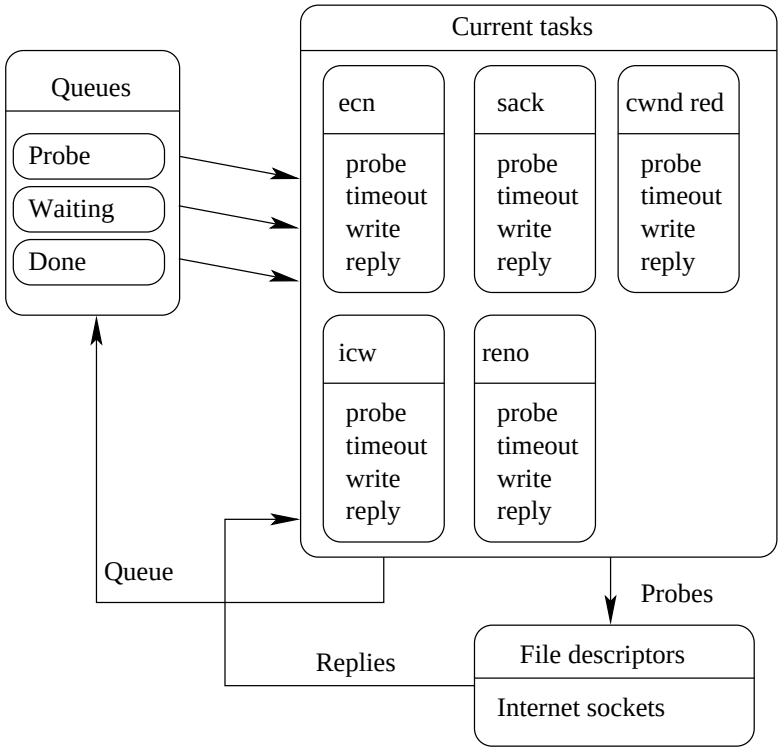


Figure 2.3: The architecture of scamper. Scamper sends probe packets to selected web servers and analyses the response packets. Scampers possible tasks include a number of methods adapted from TBIT. Key: ecn - explicit congestion notification, sack - selective acknowledgements, cwnd red - congestion window reduction resulting from loss, icw - initial congestion window, reno - flavour of TCP (tahoe, reno, new reno).

tion was also made to make the sending of SACK blocks possible, in ACK packets sent by scamper. These new scamper modules were also modified to produce warts and pcap output. The warts output made it possible to persue in-depth analyses of collected packet trace data after it was collected.

The test modules or tasks involve queueing probes and waiting for resulting replies, as illustrated in Figure 2.3. The replies are analysed to determine the behaviour of the web server involved.

2.6 Controlled environment

A facility which was set up for this research is the Controlled Environment and is illustrated in Figure 2.4. The purpose of this is to allow scamper tests to be carried out on web servers with known operating systems, to provide a situation where there are no middle boxes that might interfere, and to allow testing of scamper modules under construction, avoiding using third-party web host servers for this purpose. The controlled environment is made up of 7 computers, where 3 computers are connected together at any one time. Two of these are always the same two: scamper test and delay, and the other is one of the other five. The two permanent machines both run FreeBSD8, where one runs the chosen scamper test and the other provides a 100ms delay in each direction, to and from the web server under test. Each of the web servers uses a different operating system. These are FreeBSD8, Debian Squeeze 2.6.32, OpenBSD4.6, NetBSD5.0.2 and Windows XP SP2. Windows 7 was used for some tests where its minimum TCP data length of 536 does not cause errors, such as in the ECN test where even though

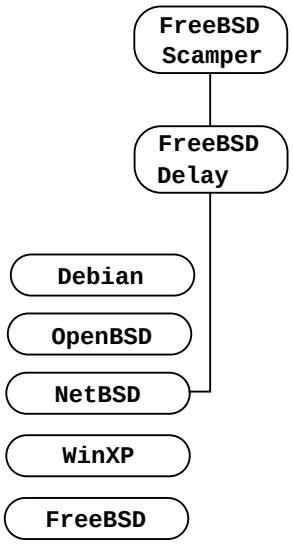


Figure 2.4: The controlled environment. The computers and their connection layout is shown, where the connection to one of five web servers can be varied. The delay is 100ms in either direction adding up to an RTT of 200ms.

capability was determined, no echo was received. Each of the web server machines runs Apache apart from the Windows machines which run IIS. Each server machine has the same large file and small file in its web service directory, available to download by HTTP. Routes have been set up on all the machines to make them behave correctly as a mini network. To connect a particular web server machine to the two permanent FreeBSD8 machines a cable is adjusted to connect only that test operating system machine.

2.7 Operating system identification

For some of this research it would be useful to identify individual web server operating systems, in order to localise particular behaviours identified in

scamper tests. NMAP is an active OS probing program which has been used in the past by the TBIT researchers [39] [33], however it was considered that NMAPs behaviour could be interpreted as a form of attack by web hosts under test. A less invasive OS classifier is p0f, which will analyse a trace of a normal TCP connection. There are however limited definitions available for the SYN/ACK mode of p0f which analyses a remote web server connected to by a web client which collects the trace data. We tested p0f on some traces collected, using wget to generate the web traffic and tcpdump to collect the packets. P0f made less than 2% predictions from these data when analysis was carried out, ruling it out as a useful classifier. Another possibility to get some server OS information unobtrusively is to collect HTTP server data from HTTP responses to HTTP 'GETs'. Though this specifies only "Apache" or no data in about 40% of cases, it still provides some useful OS information, however it often lacks detail such is specific OS version.

2.8 BIC and Cubic

Linux kernels 2.6.8 (Aug 2004) - 2.6.18 (Sep 2006) use BIC, and 2.6.19 (Nov 2006) and above use Cubic. Binary Increase Congestion control (BIC) was developed to deal with high speed networks with large delay [54] and is illustrated in Figure 2.5. It was later superseded by Cubic [46] in linux, which is simpler and less aggressive. The congestion avoidance phase of BIC involves additive increase followed by binary search and maximum probing phases. These three phases were replaced by a cubic function in Cubic TCP, as shown in Figure 2.6. When a packet loss event occurs, as inferred by

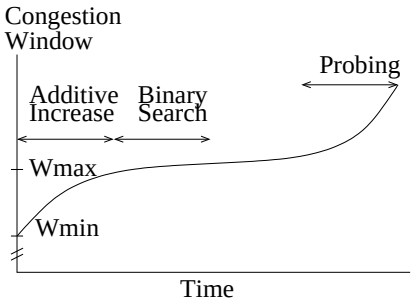


Figure 2.5: BIC congestion window control. Additive increase and binary search are shown on the approach to Wmax. Wmin is the destination of the previous window reduction. Wmax is the congestion window size at which loss last occurred.

three or more duplicate acknowledgements, the window value just prior to reduction is assigned to Wmax (cwnd maximum), and the value that it is reduced to, is assigned to Wmin (cwnd minimum). Both BIC and Cubic involve a slow approach to Wmax or previous congestion window, followed by aggressive increases or probing. In Cubic this increase is linear when far above Wmax i.e. the window increment is clamped to Smax per second. Smax is typically set to 160 [46]. The reduction is by a multiplication factor which is typically 0.8 or 0.875 [54] [46], rather than half. In both protocols Wmax is located where the flat part of the curve occurs, the point at which loss previously occurred.

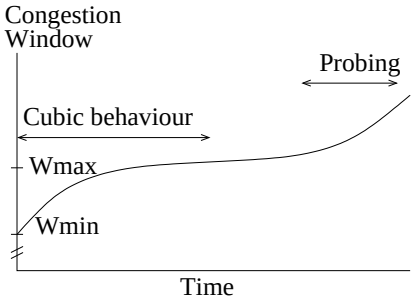


Figure 2.6: Cubic congestion window control. Cubic behaviour is shown on the approach to Wmax. Wmin is the destination of the previous window reduction. Wmax is the congestion window size at which loss last occurred.

Chapter 3
Congestion Window Behaviour

3.1 Introduction

Congestion control is a pivotal foundation of the Internet [40, 18, 22, 2, 19, 25]. It is expected that the congestion window will be halved after packet loss [4], as this behaviour prevents congestion collapse from occurring [41]. Conformance with this requirement has been measured in the past, and 2% non-conformance in a population of web servers was observed [36] in 2001 and 2004. It is of ongoing interest to monitor for possible threats to Internet stability.

3.2 Recent developments

This chapter describes developments based on the Congestion Window Halving analysis method of TBIT [36] [40]. The test measures reduction in congestion window after packet loss inferred with duplicate acknowledgements. A reduction of less than half is considered to be non compliant with RFC

2581 [4]. A fast retransmit resulting from three or more duplicate acknowledgments should not be followed by slow start but congestion window halving and congestion avoidance [20] [2]. The reason that slow start should not be used in this circumstance is that packets are still being received as is evidenced by the duplicate acknowledgements [4]. We build on an existing TBIT technique by finding a way to avoid measuring unexpected behaviour of some operating systems, which occurs after the first drop but not usually the second. The expected behaviour is shown in Figure 3.1. We are also interested in interpreting some congestion window reductions to more than half the prior congestion window as BIC or Cubic, as these are thought to constitute a significant proportion of modern web servers [18].

The development of operating systems in practice can lead to situations where there are such deviations from standards. It is also possible for some operating systems to accidentally operate outside of expected boundaries, due to programming errors, and exhibit smaller congestion window reductions or even no reduction when loss occurs.

3.3 Related work

The TBIT Congestion Window Halving test works by opening a TCP connection with a website, and then acknowledging packets received until a congestion window of 8 segments is built up, and then dropping packet 15 [40]. Duplicate acknowledgements are sent at this point, until the dropped packet is retransmitted. Retransmission should occur before an RTO interval passes from when the duplicate acknowledgements started, thus without timing out.

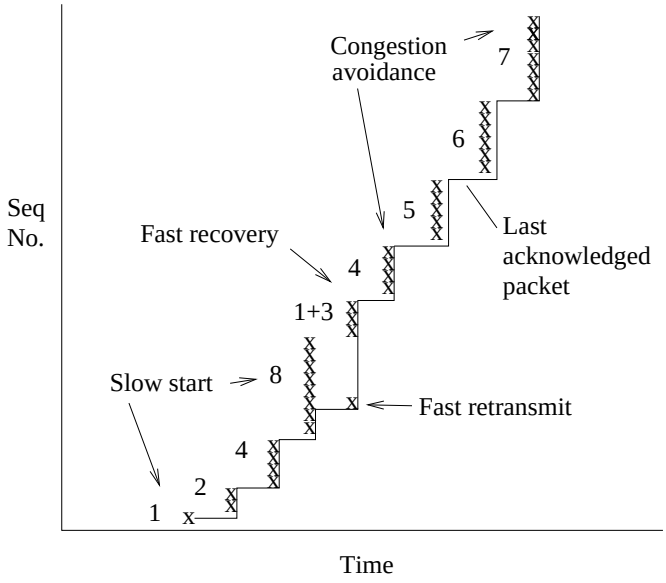


Figure 3.1: Expected behaviour of a TCP stream under packet loss inferred by three duplicate acknowledgement packets. Slow start and exponential increase is shown ending in a dropped packet. Fast recovery and fast retransmit follow which result in halving of the congestion window, and then congestion avoidance which climbs at a linear rate. This is the behaviour of Reno and New Reno TCP.

After a cumulative acknowledgement, no more acknowledgement packets are sent, and the received packets are counted. The test ends when a second retransmission of the dropped packet occurs, after the connection times out. A value of 5 packets or less is interpreted as Window Halved, otherwise the host is not compliant.

Window halving rate among webservers in US web cache logs was found in 2004 at 94% of classified servers [36]. This was a good level of compliance with the existing RFC specification [4].

3.4 Improving the test

To start with TBIT Congestion Window Halving was adapted to run under scamper. This involved rebuilding and providing the structures and services that TBIT provides to its program modules. It also involved adaptation to an events driven environment. It was still however possible to leave the module unchanged in its function.

There is some variability seen in the ICW as published by others [36], and even more so as seen in this research. The partial shift from an ICW of 2 to 4 and sometimes more makes it now more desirable to measure the congestion window immediately before the drop, rather than simply estimating it by assuming that ICW is two packets i.e. an ICW of four could mean that dropping packet 15 indicates a congestion window of 16 rather than 8. The TBIT test appears to be well suited to the 2004 environment, but the current environment has these new problems.

URL	cwnd
Linux: www.wand.net.nz/scamper...	3 segments
FreeBSD8: package.dyndns.org/m̃jl/scamper...	1 segment
Linux: www.kernel.org	3 segments

Table 3.1: Operating system and associated URL of the web server are shown in column 1. Column 2 shows measured reduced congestion window after loss as inferred by three duplicate acknowledgements. The analysis tool to obtain these results was adapted but unmodified TBIT. A result of 4 segments is expected if the congestion window reduces by half.

A small number of local websites were selected and analysed with original TBIT run under scamper. Reductions to less than or equal to half i.e. four, were observed as shown in Table 3.1. An extreme reduction down to a congestion window of one packet was observed in one case.

Because a result of one or three packets is not many packets different from four, though percentage-wise it is, it was decided to open the congestion window wider by dropping a later packet. This means that a difference of one packet has less affect on the resulting percentage of the window at the loss event. To investigate this area further the modified TBIT method was also allowed to continue after the point at which it would have been terminated. This was done by acknowledging the packets received after the fast retransmission. A small number of popular websites were selected and analysed this way, and packet flights were calculated and collected. To do this, received packets where grouped based on their timing. A gap greater than 50% RTT was considered to be a flight boundary. Furthermore as flights lost their natural grouping, each interval of one RTT was designated a flight. This latter phenomenon usually occurred later in the TCP connection, and is caused by the limited bandwidth of the connection resulting in a larger delay

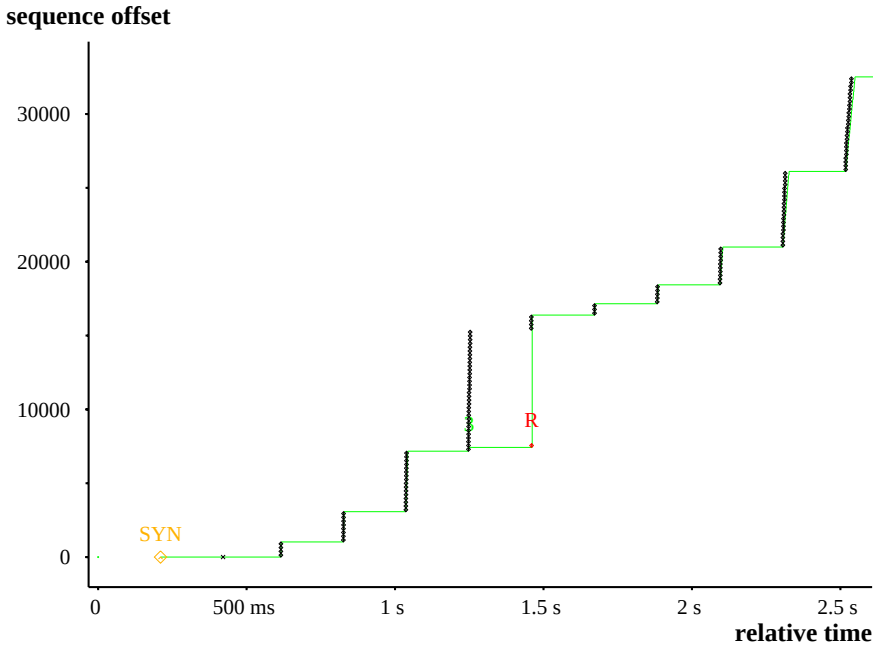


Figure 3.2: An example of loss and reduction followed by slow start. The website was www.youtube.com. Duplicate acknowledgements were limited to four. New data was ACKed when it arrived.

from packet to packet in a flight. Sometimes a reduction to a low value was followed by slow start, as illustrated in Figure 3.2. We originally got quite a high proportion of website hosts showing this behaviour, but this was when the number of duplicate acknowledgements was limited to four after the first drop. When a duplicate acknowledgement was sent for each packet received after loss, less web hosts exhibited this second slow start behaviour.

This slow start behaviour is an unexpected result so the test was repeated in a controlled environment. This was also decided partly because our attempts to identify operating systems with p0f were unsuccessful, and partly because of the possibility of middle boxes interfering with measurement. The test bed described in Section 2.6 was used to collect initial data about various

OS	Loss	Loss-4-Acks
FreeBSD 8	ss-ca	ss-ss-ca
OpenBSD 4.6	ss-ss-ca	ss-ss-ca
NetBSD 5.0.2	ss-ss-ss	ss-ss-ss
Debian Sqeeze 2.6.32	ss-ca	ss-ss-ca
Windows XP SP2	ss-ca	ss-ca

Table 3.2: Behaviour after loss for each operating system in the controlled environment. Loss behaviour is shown for where full duplicate acking is provided and then where duplicate acking is limited to 4 after the first drop. ss - slow start, ca - congestion avoidance, dash - loss event as inferred by three or more duplicate acknowledgements. A sequence of these behaviours is listed in chronological order in columns 2 and 3.

operating systems without any interference from middleboxes. The results are shown in Table 3.2. The OpenBSD and NetBSD machines went to slow start after loss signalled by duplicate ACKs. We then modified the test to drop a second packet. This resulted in expected congestion window reduction behaviour at the second drop for OpenBSD as illustrated in Figure 3.3, but not NetBSD.

The third column of Table 3.2 shows the behaviour when the test sends only 4 duplicate ACKs after the first drop, in the controlled environment. Here we see that FreeBSD and Linux were affected by the reduced duplicate ACKing, but this effect was lost when full duplicate acking was used. It is interesting that three duplicate ACKs were not sufficient in these cases to trigger normal fast retransmission behaviour, as might be expected from the specifications [4].

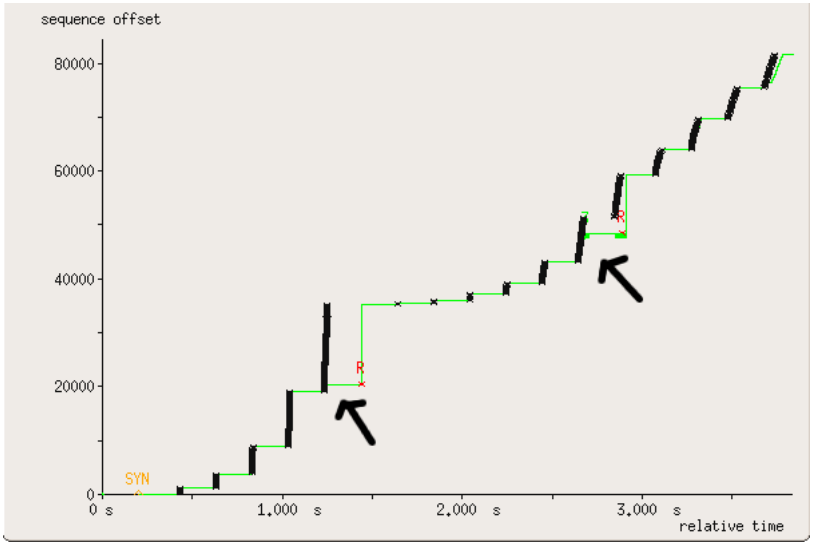


Figure 3.3: An example of loss and reduction on the second drop, then exhibiting expected behaviour. The operating system in the controlled environment was OpenBSD. Arrows show where dropped packets occurred. Two slow start sequences are observed, and normal congestion window reduction is observed subsequent to these.

3.5 Method

The modified method also analyses congestion window reduction after packet loss from the TCP data stream. This method is similar to TBIT but takes steps to avoid measuring anomalous slow start behaviour which is sometimes encountered. It also attempts to get a profile of congestion windows during a trace rather than just measurement before and after loss. Though the final measurement is similar, attempts are made to categorise results into more groups than just greater than or less than half the original congestion window. This is because of the existence BIC and CUBIC TCP which reduce their congestion windows to 80%, when loss occurs as indicated by three or more duplicate acknowledgements. We also distinguish reduction to much less than half the congestion window, as the first drop in some cases reverts to slow start. Furthermore we attempt to measure the congestion window immediately prior to the drop event, rather than just calculate what it should be after a set number of packets. Also, the congestion window was opened wider than in the TBIT test, in order to improve accuracy of the reduction measurement. A test where window reduction could be measured after a first or second drop, depending on presence of repeated slow start behaviour was a logical modification.

The modified method is called Congestion Window Reduction Analysis. Four main changes to the original method have been made. They are detailed in the following subsections.

3.5.1 Second slow start

It was decided to measure congestion window change after a second drop when a second slow start occurred, in order to observe and analyse conventional fast retransmission behaviour, because in some cases fast retransmission and an associated window reduction was seen after a second drop when not seen after the first.

3.5.2 Congestion Window Profiles

Where there is a reasonable length RTT, there is time for a full congestion window of packets to be sent bunched within the RTT, and then a wait follows for acknowledgement before transmission can continue. The flights of packets produced this way are bounded when a wait for acknowledgement occurs, but the flow of packets is not stopped, allowing many subsequent flights to be measured without a loss of any kind occurring. This is not a commonly used procedure, but there are a small number of instances of its use [18, 52, 51, 42]. One method of confirming this interpretation of flights is to count the packets from when the drop occurs until when the retransmission occurs, and to compare this to the flight in which the drop occurs. Beyond this it is necessary, as in the TBIT method, to stop ACKing and count the packets sent until a retransmission timeout (RTO) occurs, in order to measure the congestion window at one place only.

Analysis of flights is used to obtain a profile of congestion windows. It is used in the Congestion Window Reduction Analysis method to estimate congestion windows without the need to stop acknowledging packets and

wait for a timeout. This is a desired outcome because it is then possible to check for expected congestion avoidance behaviour after retransmission, and to observe the congestion window before and after the drop.

3.5.3 Increased Congestion Window Size

The TBIT method dropped packet 15, and the window at this point was assumed to be eight packets. No attempt was made to measure this, only the size of the final window, as a percentage of eight. Because we wished to distinguish BIC and CUBIC from regular TCP and to also distinguish the anomalous slow start behaviour, we chose to increase the size of the congestion window before introducing loss. This meant allowing more packets to be sent before dropping a packet. Because we wished to measure congestion window before and after a second loss the initial congestion window needed to be bigger still, as the Slow Start Threshold (ssThresh) is reduced by the first drop, which is at packet 50.

3.5.4 Increased MSS

The TBIT method used a MSS of 100 bytes. In our initial testing we encountered some operating systems which would not accept a value as low as 100 bytes, so MSS was increased to 256 bytes. The TBIT researchers suggested that MSS should be optimised for running this kind of test [36]. This change reduced the error rate of the test, and did not seem to have any other effect, although we had to ensure that we selected webservers with sufficient data to supply the larger sized packets.

3.6 Data collection

In order to generate a data file for this test a perl crawler program was used to gather IP addresses and data size of the websites. This was carried out on 30000 websites. Another perl program was used to filter out websites with data size less than 75000 and repeated IP addresses, this left 6784 websites, 23%. A driver was used to run the test and carry out up to three retries if 'missed drop' or 'no retransmit' errors occurred. Drops were specified at packets 50 and 170 because the congestion window needed to be allowed to open wide for the reduction measurement, the second drop if required, also needed to allow for a large flight in which the first drop occurred.

The resulting window reductions were categorised as 'Cwnd less than 20' where the window reduced to less than 20%, 'Regular TCP' where the range was 20% to 70%, 'BIC or Cubic' where the range was 70% to 120%, and 'cwnd increase' where there was an increase above 120%. The same categories were applied to the second drop if a second slow start occurred. These boundaries are based on the expected results plus or minus an error margin.

3.7 Results

Congestion window reduction analysis was run in the controlled environment, and the results for the five operating systems are shown in Table 3.3. FreeBSD and Windows XP exhibited Regular TCP and Linux BIC or Cubic. Second slow starts were seen for OpenBSD and NetBSD, and Regular TCP was exhibited by the former whereas a third slow start was seen for the later.

OS	%cwnd	Conclusion
FreeBSD8	43	Regular TCP
OpenBSD4.6	56	Regular TCP
NetBSD5.0.2	12	Tahoe (Slow Start)
Debian Squeeze2.6.32	92	BIC or Cubic
Windows XP SP2	66	Regular TCP

Table 3.3: Congestion window reduction and probable meaning. Column 1 shows the operating system of the analysed web server machine, column 2 shows the window after loss as percentage of window before loss, and column 3 shows the likely interpretation of column 2. The interpretation/conclusion is based on %cwnd percentage ranges. The OpenBSD and NetBSD values are both measured after a second slow start.

The results from the window reduction test are shown in Table 3.4. The error categories are as follows. 'No TCP connection' or 'Early reset' indicate that the TCP connection failed. 'No data response' indicates that an early FIN packet was received. 'HTTP error' indicates that HTTP '200' OK was not received in the data response. 'Drop' indicates that an unwanted packet drop occurred. 'Not enough packets' indicates that insufficient data packets were available in the packet stream; that is, an early FIN packet was received. 'MSS error' indicates that a packet larger than the negotiated MSS was received. 'No data' indicates that the test timed out three times waiting for data. 'Drop not found' and 'Not enough flights' indicate that the flight containing the drop was not able to be identified, this occurs when a large number of very small flights are found and a normal increase in congestion window is not seen. 'Late retransmit' indicates that the retransmission occurs later than the third flight after the drop. 'Net retry errors' indicates the remaining errors after the repeats of 'Missed drop' and 'No retransmit', a combination of these two categories. 'Missed drop' indicates that the packet

to be dropped was not received and failed to trigger the drop mechanism. 'No retransmit' usually results from reordering that defeats the drop mechanism. These two latter errors resulted in the driver retrying the test up to three times. This proved effective as only 289 errors remained after the retries occurred. The overall error rate was low compared to previous studies in 2004 [36], 22% compared to 64%. The major contributors to the previous test data were 'Not enough packets' and 'HTTP error', and the reason we did not see so much of these was that we checked our URL data file for validity and data quantity available. The test could be modified to eliminate MSS errors by allowing packets larger than MSS and further reduce the error rate.

Medina *et al.* [36] reported 94% of classified servers as having reduced their congestion window by at least 50% in 2004. This is similar to combining 'Cwnd less than 20' and 'Regular TCP' categories, which correspond to 37% of classified servers. This includes the final window range of 0% to 70% making 37% of classified servers an upper bound for 50% reduction. The main group is 'BIC and Cubic' at 56% of classified servers, and there is a small number that unexpectedly increase their congestion window.

There is only a smaller number that show a second slow start, and the largest group of these exhibit a possible third slow start and a smaller number exhibit 'regular TCP'. Smaller still is a group of these that exhibits BIC or Cubic.

Figure 3.4 shows the cumulative distribution function of percent congestion window change after loss. There is a distinct region of reduction to the range of 2% to 15% which accounts for 10% of cases. There is then a steady gradient from 50% to 80% change. After this there is another steady steeper

Result	Count	Percent
No TCP connection	44	0.7%
Early reset	16	0.2%
No data response to TX request	62	0.9%
HTTP error	125	1.8%
Drop	19	0.3%
Not enough packets	201	3.0%
MSS error	404	6.0%
No data	7	0.1%
Drop not found	50	0.7%
Not enough flights	6	0.1%
Late retransmit	265	3.9%
Net retry errors	289	4.3%
Errors	-	*22.0%*
Cwnd less than 20	181	2.7%
Regular TCP	1048	15.4%
BIC or CUBIC	2830	41.7%
Increased cwnd	360	5.3%
2nd SS, Cwnd less than 20	448	6.6%
2nd SS, Regular TCP	263	3.9%
2nd SS, BIC or CUBIC	133	2.0%
2nd SS, Increased cwnd	33	0.5%
total IPs	6784	-
Missed drop	268	-
No retransmit	1117	-
Retry after error	1096	-

Table 3.4: Results category counts and percentages for the window reduction test carried out on 04/12/2010. Error counts are grouped at the top of the table, and the total error rate is shown beneath these. The four window reduction ranges are shown twice once for the first loss, and then a second group which signify that reduction was measured after a second loss event, which occurred after a second slow start.

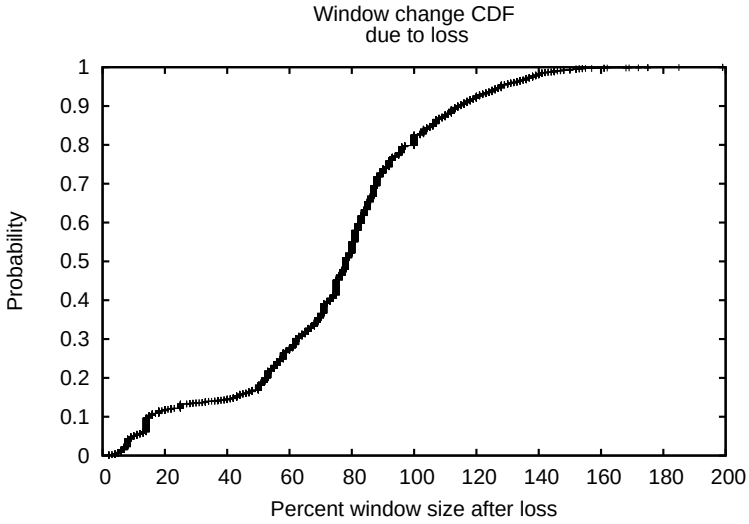


Figure 3.4: Cumulative distribution function for percent congestion window change due to loss. The x axis is the window size after loss, as a percentage of window size at the time of packet drop and the y axis is the cumulative probability.

gradient from 80% to 100%. These different gradients indicate distinct regions, which are similar to the regions that were chosen empirically, apart from the region from 100% through to 120% which gradually levels out. It can also be seen that only 20% of webservers reduce their congestion window by half as required by the original specification. The main contributors to this changed situation appear to be a shift to BIC or Cubic and a group of machines that fail to make a reduction. There also appears to be a group slightly above 50% of prior window, which is within an error margin.

3.8 Conclusions

There are two small second slow start categories that relate to behaviour similar to OpenBSD and NetBSD. Extra window behaviour information has been successfully gathered by extending analysis to second drops when appropriate.

Congestion behaviour affects Internet stability, and implementations involve a trade-off between stability and performance. We found a swing in behaviour from emphasis on stability to greater emphasis on performance. There has been significant migration of systems exhibiting regular TCP and reduction to congestion windows smaller than half, to BIC and Cubic style algorithm based systems. We also see a small amount of aberrant behaviour where the congestion window is observed to increase. In all these cases retransmission of the dropped packets was confirmed.

Chapter 4
Explicit Congestion Notification

4.1 Introduction

ECN is a TCP enhancement designed to control congestion without the need for packet loss [45]. For this reason, if ECN is implemented it is likely to result in improved Internet performance with reduced packet loss. It is therefore desirable to know the extent to which ECN has been successfully implemented on the Internet, and compare our measures with earlier measurements [36].

This chapter describes ECN and how its implementation has been measured before. Then our ECN test is described along with details of data collection. We find a steady increase in successful ECN implentation over several months, some indications that many machines that successfully implement ECN are Linux systems and that anomalous CWR packets are sent by some of these successful servers.

4.2 Explicit Congestion Notification

Regular TCP congestion control involves the detection of packet loss as a signal to reduce the congestion window, though not all loss is due to congestion e.g. wireless where loss due to transmission errors is a common occurrence [49]. Though ECN makes use of end to end communication in its function, it is dependent on upgraded network infrastructure experiencing congestion to mark packets instead of dropping them. This means that not only must end hosts be upgraded to ECN capability but also network routers. A TCP flow may use ECN if it was negotiated by the two end hosts at connection time.

ECN makes use of the least significant two bits of the IPv4 Type Of Service field [45] or the least significant two bits of the IPv6 Traffic Class byte, and the most significant two bits of the TCP flags field. When ECN is used with TCP, fields are used to signal ECN Echo (ECE) and Congestion Window Reduced (CWR), while the IP fields are used to signal ECN Capable Transport (ECT) and Congestion Experienced (CE).

00 - Non ECN-Capable Transport - Non-ECT

10 - ECN Capable Transport - ECT(0), (preferred when used with TCP [45])

01 - ECN Capable Transport - ECT(1)

11 - Congestion Encountered - CE

The implementation of ECN in IP is insufficient for use without further implementation in a transport protocol like TCP, because there is no signal for an echo nor is there a signal to acknowledge the echo once it has been responded to. IP does however take care of carrying packet CE marks.

Once packets are marked as having encountered congestion at the IP level, they must be echoed back to the sender by the receiver. The ECE flag in the TCP header is used as this ECN echo signal. The TCP CWR flag is then used to signal that the sender has reduced its congestion window and the echo may be acknowledged. If the echo is not cancelled the echo continues to be sent. Apart from SYN packets and ECE ACK packets, only data packets contain ECN flags [28]. The use of these flags is shown in Figures 4.1, 4.2 and 4.3.

A TCP connection which negotiates ECN sends a SYN packet with the ECE and CWR flags set. If ECN is negotiated the SYN ACK packet has the ECE flag only set.

This chapter examines whether a selection of web servers from the Alexa list are able to negotiate ECN, and then whether they are able to echo a marked packet. These are the key features of successful ECN implementation.

4.3 Related work

There is a TBIT test used to determine ECN success of a webserver [36] [39]. In order to determine if ECN negotiation results in a failed connection, a non ECN connection is attempted before attempting an ECN connection and the ECN test. The ECN test starts by sending an ECN SYN to the web server. If no response is received a further two SYN packets may be sent. If a SYN ACK packet is received the test continues. The SYN ACK packet is checked for the presence of an ECE flag and the absence of CWR as confirmation that ECN was negotiated. Regardless of this last outcome an HTTP request

is sent with the IP header ECN field set to CE to mark the packet as though a router had marked it. If an ACK is received the ECE flag is checked for. If this is not present the web server does not support ECN.

The test we use differs from this test in that non ECN negotiation occurs only if ECN negotiation fails, the test only continues after negotiation if the correct SYN ACK ECE packet is received, and IPv6 webservers are tested.

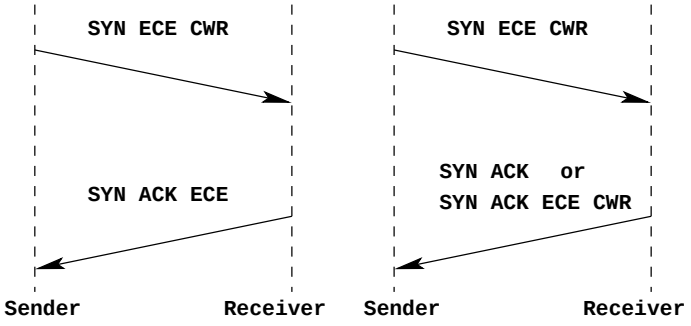
ECN success rate among webservers in US web cache logs was found in 2000 at 0.1% and 2004 at 0.5%. In this time the rate of failed connections, only with ECN, dropped from 9% to 1%, and error rates were below 10%. We see here the beginning of an upward trend in the correct implementation of ECN, and reduction in the rate of connection failure due to ECN negotiation attempts.

In a previous project in September 2009 [14] I measured the rates of ECN success for IPv4 at 2.2% and IPv6 at 2.6%, using the top 400 and top 300 websites respectively from Alexa.org. In this study repeated measurements over time are carried out using larger data sets of 10000 IPv4 addresses, and a driver is used to automatically carry out null tests when necessary. Higher levels of successful ECN implementation are seen.

4.4 Method

Scamper TBIT is similar in its behaviour to original TBIT, however a driver is used to run ECN tests and if this fails to connect or is reset, only then is a null test run. The null test checks to see if all SYN packets are unacknowledged or just ones negotiating ECN. This null test simply negotiates a TCP connection

without ECN, then requests data, and receives and acknowledges data until the download ends normally. The scamper TBIT test also completes the website download in order to appear like a normal browser. Figures 4.1- 4.2 show the packet exchange structure of the ECN test. Scamper TBIT does not send an HTTP request unless ECN is successfully negotiated.



(a) Successful ECN negotiation.

(b) Failed ECN negotiation.

Figure 4.1: Subfigures (a) and (b) show successful and failed TCP ECN connections. A successful connection requires a SYN ACK packet with just one ECN flag, ECE.

4.5 Data collection

The list of one million Alexa most popular websites was downloaded and used as input for a perl web crawler program to generate a data file for the ECN test, containing the IP address with the URL to use. Data for the top ten thousand websites was collected in this way. This list was then used with the ECN scamper driver and a scamper daemon, to carry out the ECN test on these websites. A script was used to repeat this analysis every two weeks to gather a profile over time.

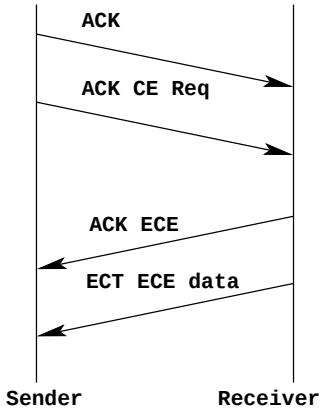


Figure 4.2: The sending of an ACK followed by a marked HTTP request packet is shown. Following this an ACK packet and data packet each carrying an echo are returned by the receiver.

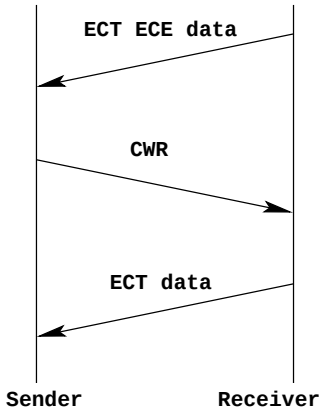


Figure 4.3: A data packet carrying an echo is shown, followed by a packet with the CWR flag. After this there are no more echos.

OS	ECN behaviour
FreeBSD 8	Not ECN capable
FreeBSD 8 ECN enabled	ECN no echo
OpenBSD 4.6	Not ECN capable
NetBSD 5.0.2	Not ECN capable
Debian Squeeze 2.6.32	ECN echo
Windows XP SP2	Not ECN capable
Windows 7	Not ECN capable
Windows 7 ECN enabled	ECN no echo

Table 4.1: Type of ECN behaviour observed for each operating system in the controlled environment. 'Not ECN capable' means that ECN could not be negotiated. 'ECN no echo' means that though ECN was negotiated, the mark was not echoed. 'ECN echo' means that the test was successful and the mark was echoed back.

The ECN test was also carried out on the machines in the controlled environment. These data were used to characterise the operating systems on these machines in the absence of middle boxes.

Another web crawler was run on the full set of one million websites on 12/1/2011, which found webservers via their IPv4 addresses which also had IPv6 addresses. After collecting these IPv4, IPv6 addresses and URLs the data set was further refined to contain only globally routable addresses. This includes the IPv4 addresses along with 2000::/3 IPv6 addresses and excludes 2002::/16 IPv6 addresses. This resulted in a relatively small data file with 2964 addresses, as the number of webservers with globally routable IPv6 addresses is still small.

result	2004	count	percent
Classified Servers	95%		*97.5%*
ECN incapable	93%	9272	88.7%
ECN capable	2.1%		*8.8%*
ECN success	0.5%	853	8.2%
No ECE (echo)	1.5%	66	0.6%
Bad SYN ACK	0.2%	161	1.5%
No data packets	0%	1	0.0%
No TCP connection	3.8%	93	0.9%
Early TCP reset		7	0.1%
total		10453	
null			
null-success	1%	46	0.4%
Early TCP reset		4	0.0%
Later TCP reset		6	0.1%
No data packets		2	0.0%
No TCP connection	2.8%	42	0.4%
total		100	

Table 4.2: These are the results from an ECN test of 10453 websites on the 01/11/2010. TBIT results from 2004 [36] are also shown. There is a count of websites and a percent value, for each result type.

4.6 Results

Debian Squeeze Linux was the only OS of those tested which showed successful ECN behaviour, as shown in Table 4.1. ECN enabled FreeBSD 8 and Windows 7 were ECN capable but gave no echo, and the others could not negotiate ECN.

The results in Table 4.2 are typical of all the data points in terms of error rates. We see that the null test was run 100 times (93 + 7) and half of these resulted in successful connection. This indicates that 0.4% of machines reject ECN connection while accepting regular connections, quite a low level

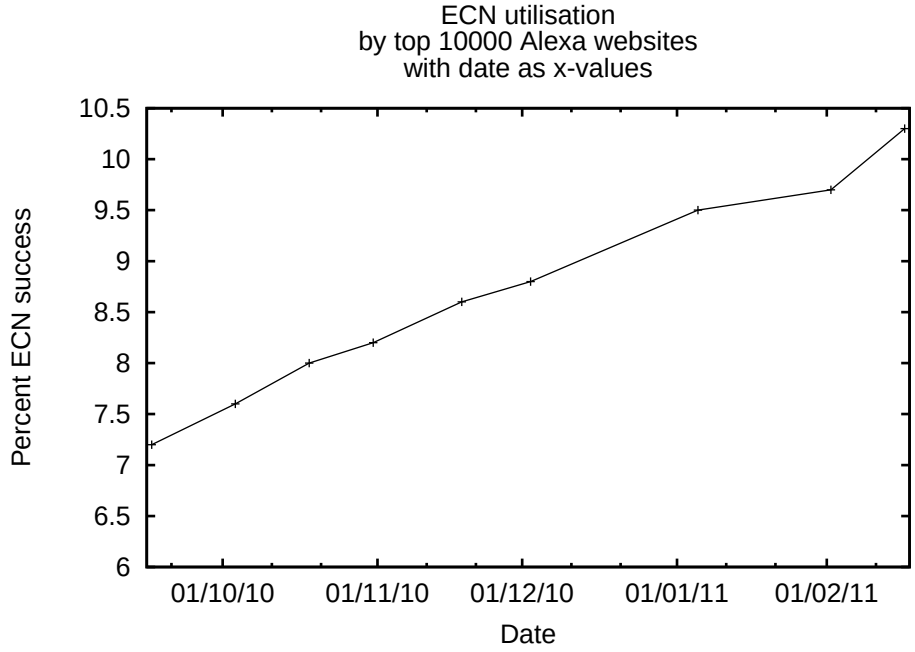


Figure 4.4: The success rate of the ECN test versus date is shown. The data set was the top 10000 Alexa popular websites, as downloaded from the Alexa website on the dates where there are data points plotted.

compared to 2000 [36] when it was 9%. All other error counts are quite small.

The meaning of the other errors is as follows. 'No TCP connection' means that no connection was achieved, 'Early TCP reset' is similar but indicates that a reset packet was received. 'Bad SYN ACK' means that SYN ACK ECE CWR was received, and 'No data packets' means that no data was received in response to the HTTP request. The null test error messages have a similar meaning to those in the ECN tests.

Of classified server results 'No ECE' means that ECN was negotiated but there was no echo, 'ECN incapable' indicates that ECN could not be negotiated and 'ECN success' indicates that an echo was received.

date	cwr
Fri 17 sep	24.6%
Mon 04 oct	18.0%
Tue 19 oct	27.0%
Mon 01 nov	5.5%

Table 4.3: This table shows the percentage of successful traces where one or more CWR packets was received from the webserver, on the given date.

category	17 Sep	4 Oct	19 Oct	1 Nov
null+misc	190	189	216	226
Apache	241	248	254	263
nginx	153	166	172	182

Table 4.4: This table shows the number of successful ECN connections in a common HTTP server category, on the given date. There is an upward trend in each case, over time.

Figure 4.4 shows a steady increase in ECN success rate of on average 0.25% every two weeks, over several months. This implies that as operating systems are upgraded, that some of the new replacement systems successfully implement ECN. Additionally it could imply that System Administrators are generally more inclined to have ECN turned on than before.

Table 4.3 shows frequency of successful ECN traces where CWR packets were received from the webserver. On all these dates >95% of these CWR packets occurred in traces where 3 duplicate ACKs occurred. Because of this it might be expected to see a drop in percentage with CWR after the first date as this is when a patch to deal with packet reordering was written and installed, which prevented reordering from being treated as loss. These CWR packets are unexpected, and it is difficult to see why they should be there as there are no ECN echos being sent to the server, from the measurement host.

Server information was gathered from the warts data files where available, and it was noted that the general indication was that the ECN successes where made up of mostly Linux webservers. Server information specified the type of Linux but not the version, and the count for this did not vary significantly from 85 hosts. There were a small number of Microsoft Windows machines where ECN had been activated. Table 4.4 shows several large categories of ECN successes, 'null+misc' are traces were no server information was provided or a rarely seen specification was given, 'Apache' and 'Nginx' are likely to be mostly Linux systems in addition to the specifically identified ones. These three categories showed increases over time, as the number of successes increased.

The Linux kernel releases that first enabled ECN by default were released in 2006. The website news.netcraft.com reported that 60% of webservers use Apache, 21% Microsoft IIS and 7.5% nginx. A value less than the sum of Apache and nginx (68%) is the proportion of Linux servers, which is likely to be a ceiling value for ECN.

The results in Table 4.5 show the IPv4 and IPv6 ECN test results for the same but small set of webservers. It seems likely that one would expect similar results for IPv4 and IPv6 on the same machine. The IPv6 successes are however slightly higher than IPv4 and both are more than double what was seen in the IPv4 only test (9.7% success) carried out at the same time. On the other hand 'ECN capable' is slightly higher for IPv4, and the difference is mostly caused by the also higher 'no ECE' result. Possible causes of these differences include IPv4 echos being cleared or stopped by some middleboxes, operating system specific differences between ECN for IPv4 vs IPv6, and the

result	count 4	percent 4	count 6	percent 6
Classified Servers		*98.9%*		*85.3%*
ECN incapable	1096	74.1%	931	62.7%
ECN capable		*24.8%*		*22.6%*
ECN success	288	19.5%	325	21.9%
No ECE (echo)	79	5.3%	10	0.7%
Bad SYN ACK	1	0.1%	2	0.1%
No data packets	0	0.0%	0	0.0%
No TCP connection	14	0.9%	179	12.1%
Early TCP reset	2	0.1%	38	2.6%
total	1480		1485	
null				
null-success	2	0.1%	6	0.4%
Early TCP reset	2	0.1%	38	2.6%
No data packets	1	0.1%	0	0.0%
No TCP connection	11	0.7%	173	11.6%
total	16		217	

Table 4.5: These are the results from an ECN test of 1480 websites on the 02/02/2011. There is a count of websites and a percent value, for each result type. This is repeated for each of IPv4 and IPv6, where there is a suffix of 4 or 6 respectively.

larger number of failed connections for IPv6 could skew the results if there were a bias towards 'no ECE' operating systems failing to connect IPv6.

Possible explanations for the higher success rates than in the IPv4 only test include, the small size of the data set available for this test, bias introduced by using a higher proportion of less popular webservers and, the likelihood that webserver machines set up for IPv6 on the Internet use more modern ECN capable operating systems.

4.7 Conclusions

The successful implementation of ECN is showing steady improvement, however there are still some out of place events (e.g. CWR packets) which occur even in successful implementations. At the present rate of improvement it will improve about 6% per year, however this may improve as operating systems updates that fix ECN are introduced. Based on my earlier results in September 2009 of ECN success less than 3%, this rate is a recent occurrence and could be increasing.

One of the hurdles has reduced and that is the proportion of ECN connections that fail, while normal connections succeed. This factor put OS designers off activating ECN in their operating systems, because of the increased Internet access failure rates incurred. Middleboxes clearing ECN flags or dropping ECN packets were believed to be partly to blame for this problem, and the prevalence of this problem appears to have abated.

It is also clear that an improved method of identifying operating systems would be useful, as this would improve the quality of feedback on the

performance of the Internet that can be provided and the operating system specificity of such information.

It is also concluded that webservers running via IPv6 are more likely to be ECN successful than the average webserver, and that corresponding IPv4 ECN is more likely to fail to echo when ECN is capable.

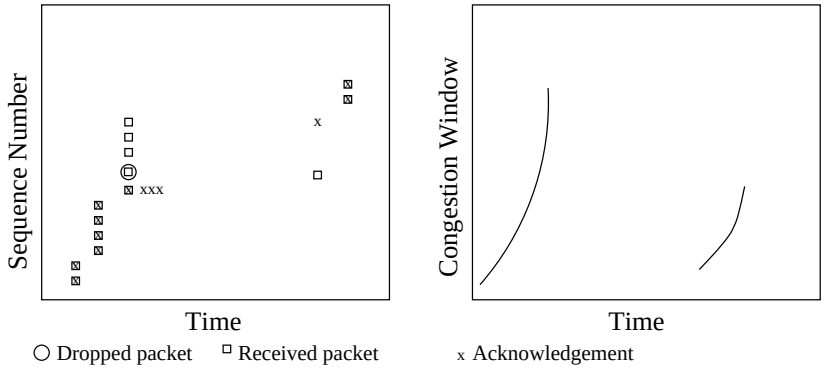
Chapter 5
Reaction to loss: Tahoe, Reno and New Reno

5.1 Introduction

Tahoe, Reno and New Reno are algorithms of TCP that operate in the absence of SACK to react to packet loss as inferred by three duplicate ACKs and control congestion. The points of difference include whether RTOs occur, whether packets are unnecessarily retransmitted and whether multiple losses can occur without these events occurring. These characteristics affect the performance of TCP when congestion occurs and as such are useful to evaluate. Furthermore these algorithms are of interest as there are Internet systems that use them, not all systems are SACK capable, and feedback on Internet behaviour is part of making developmental progress. In this chapter the prevalence of these modes of TCP are measured and the likely impact on performance is assessed.

5.2 TCP reaction to loss

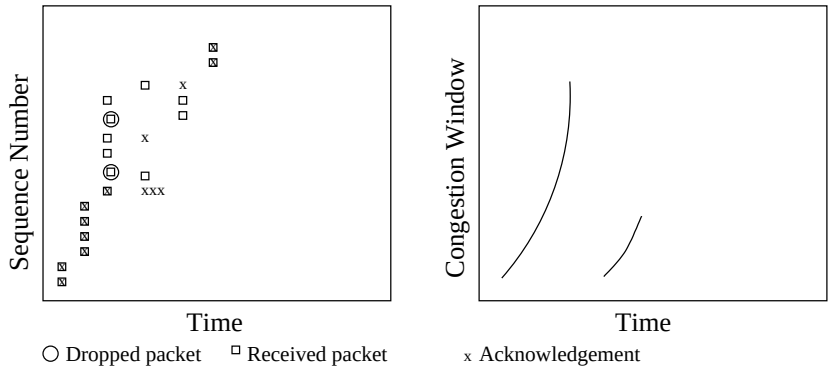
Tahoe and Reno only occur in older operating systems and have been reducing in prevalence [36]. New Reno behaviour is emulated by some newer TCP varieties such as BIC and CUBIC [46] [54].



(a) Tahoe packet trace with drop. (b) Tahoe congestion window behaviour.

Figure 5.1: Figures (a) and (b) show the trace of packet flights and the changing congestion window for Tahoe TCP. A second slow start is observed after packet loss as inferred by three duplicate acknowledgements, and an RTO.

There are a number of TCP behaviours in response to loss. Tahoe reacts to loss inferred by three duplicate acknowledgements by treating it the same as a timeout, as illustrated in Figure 5.1. The ssthresh variable is set to half the congestion window as above, initial window is set to one or two segments, and conventional slow start follows. There are implementations of Tahoe that have fast retransmit, which is distinguished from other fast retransmit algorithms by the fact that it transmits an extra packet that was not previously dropped [39] as illustrated in Figure 5.2. Reno differs from Tahoe in that it includes the fast recovery algorithm and fast retransmit [15]. In Reno fast

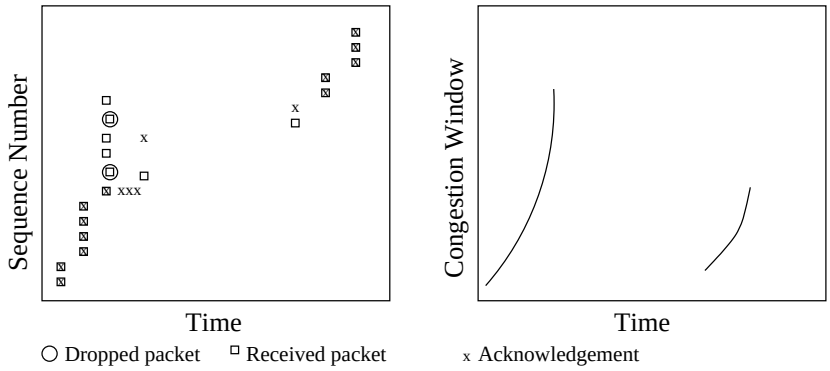


(a) Tahoe FR packet trace with drop.

(b) Tahoe FR congestion window behaviour.

Figure 5.2: Figures (a) and (b) show the trace of packet flights and the changing congestion window for Tahoe with fast retransmit TCP. Fast retransmission is observed after packet loss as inferred by three duplicate acknowledgements. An extra packet is retransmitted and a second slow start follows.

recovery involves halving the congestion window and waiting for a cumulative acknowledgement of all unacknowledged packets. After a cumulative acknowledgement, TCP continues with congestion avoidance. If a second packet is dropped the first is still fast retransmitted but the second causes a timeout. This leads back to slow start and thus Reno does not perform well when multiple losses occur, as illustrated in Figure 5.3. This is as might be expected as Reno was optimised for dealing with single packet drops [16]. In New Reno each duplicate acknowledgement results in transmission of a further packet during recovery [20]. For every progressing acknowledgement that is not fully cumulative, the next unacknowledged packet is sent. New Reno does not timeout when two packets are dropped but deals with them in this way, provided there are acknowledgements arriving, as illustrated in Figure 5.4. These drops result in a halving of the congestion window and



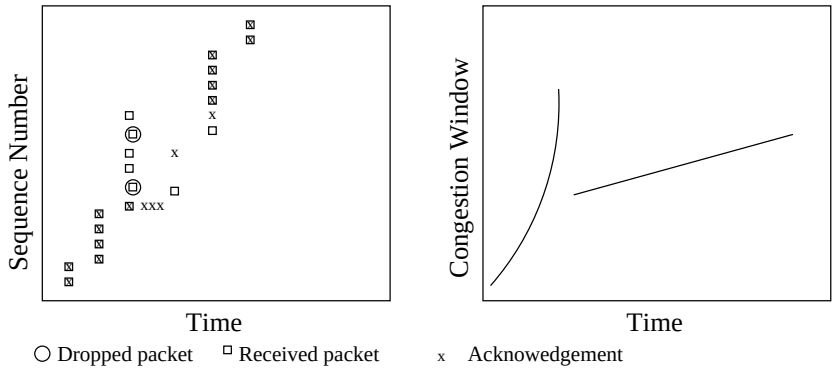
(a) Reno packet trace with drops. (b) Reno congestion window behaviour.

Figure 5.3: Figures (a) and (b) show the trace of packet flights and the changing congestion window for Reno TCP. A second slow start is observed after packet loss as inferred by three duplicate acknowledgements.

congestion avoidance. Reno evolved from Tahoe and New Reno from Reno as a result of the need to improve throughput under congestion conditions, as much as is permissible without further adding to congestion [37].

5.3 Related work (TBIT)

TBIT starts by connecting to the remote web server using the three packet handshake, where MSS is set to a small value such as 100 bytes and a receiver window of 5 times this is set. Then an HTTP get request is sent for the webpage. TBIT acknowledges the data packets received from the remote server. Packet 13 is then dropped and the ACKs for packets 14 and 15 are duplicate ACKs for packet 12. Packet 16 is dropped and subsequent acknowledgement occurs as appropriate. The connection is closed after 25 data packets have been received.



(a) New Reno packet trace with drops.

(b) New Reno congestion window behaviour.

Figure 5.4: Figures (a) and (b) show the trace of packet flights and the changing congestion window for New Reno TCP. Congestion avoidance is observed after packet loss as inferred by three duplicate acknowledgements.

NewReno is detected if there is a fast retransmit for packet 13, no RTOs and no retransmission of packet 17. Here the first packet dropped is fast retransmitted, packet 17 was not dropped and is the one after the second packet that was dropped and retransmitted. RenoPlus (or aggressive fast retransmit) is detected when there are no RTOs for packet 13 and 16, there is transmission of additional packets between the retransmissions of packets 13 and 16, and there are no unnecessary retransmissions. RenoPlus is a step toward New Reno from Reno. Reno is detected when there is a fast retransmission of packet 13, there is an RTO for packet 16, and no unnecessary retransmission of packet 17. Here the first packet which was dropped was fast retransmitted and the second drop timed out. The unnecessary retransmission of packet 17, the packet after the second drop, which did not occur here is a feature of Tahoe. Tahoe with fast retransmit is detected when there is fast retransmission of packet 13, and unnecessary retransmission of packet

17. Here the dropped packet is fast retransmitted, an unnecessary packet is retransmitted, the next packet is transmitted and then the second packet is retransmitted. Tahoe with no fast retransmit is detected when there is an RTO for packet 13 and unnecessary retransmission of packet 17. Here the first dropped packet is retransmitted after an RTO, after an RTT the second retransmission and the extra retransmission are sent. Aggressive fast retransmit is detected when there are no RTOs, there are more than 3 retransmissions, and unnecessary retransmission of packets 14 and 17. This algorithm does not time out but does retransmit unnecessary packets. Aggressive Tahoe with no fast retransmit is detected when there is an RTO for packet 13, there is no RTO for packet 16, and there are more than 2 retransmissions including that of packet 14. Once again the aggressive version involves extra retransmissions.

5.4 Method

The scamper test used was similar to the original TBIT method, however a driver was used to repeat the test up to three times if unwanted drops or reordering occurred in the trace. Previously TBIT tests where repeated five times requiring agreement of three [39], and in later work [36] involving larger data sets each test was run only once.

The top 10000 Alexa websites were used as data, a subset of the top 1 million. The Reno test was run on the 22/11/2010 on a machine outside the University of Waikato's firewall.

The initial version of the Reno test resulted in 9% of cases where the test failed because of varying packet size, and 35% of cases where out of order packets caused failure. The program was rewritten to deal with varying packet size, and a driver was written to rerun the test up to three times when reordering was encountered. These changes reduced the error rate to 25% due to reordering and 0% due to packet size variation. This testing was carried out on a sample size of 1000 randomly selected from the top 10000 Alexa websites.

This modification had the side affect that reordering was included with drops as a failed result. A further modification was made to allow variable packet size and reordering to occur. The difficulty was that the program must predict a packet number for each packet it receives. This includes the case where a packet is received before its predecessor. In order to make a prediction the size of the packet before a gap is used to predict the size of the gap, either one or two packets worth. If the prediction did not fit the size of the gap the test failed. This modification reduced the error rate, though there were still errors due to excessive packet size variation, retransmission of packets with different sequence numbers and repeated drops.

5.5 Results

The error rate is lower than for the measurements made in 2004 [36], as shown in Table 5.1. This breaks down into the following categories. 'No TCP connection' and 'Early reset' indicate a failed TCP connection. This was similar to 2004. 'No data response to TX request' indicates that the

Result	2004	Count	Percent
Classified servers	33%	-	*78.3%*
Reno	5%	774	7.7%
New Reno	25%	6690	66.6%
Tahoe with fast retransmit	1.2%	290	2.9%
Tahoe (no fast retransmit)	1.4%	18	0.2%
Aggressive Tahoe no FR	0%	1	0.0%
Reno, Aggr FR	0.2%	6	0.1%
Uncategorised	0.4%	77	0.8%
Errors	53%	-	*19.1%*
No TCP connection	2.6%	134	1.2%
Early reset		330	3.3%
No data response to TX request	4%	98	1.0%
TCP Error		379	3.8%
Extra retransmits		29	0.3%
Not enough packets	27%	851	8.5%
15 drop after 12 RT		78	0.8%
12 drop after 15 drop		16	0.2%
Classified but ignored		-	*2.7%*
net unwanted drops etc.		272	2.7%
total IPs		10043	-
Unwanted drops		527	-
Packets not sequential		597	-
Packet not seen before		2024	-
Retries after unwanted drops etc.		2876	-

Table 5.1: Results category counts and percentages for the Reno test carried out on 22/11/2010. TBIT results from 2004 [36] are also shown. The counts adding up to 'total IPs' are shown above that line and the number with 'Unwanted drops', 'Packets not sequential','Packet not seen before ' and the number of retries are shown below that line. If any of these three errors occurs then the test is rerun up to a total of three times.

connection timed out three times waiting to receive data. This was one quarter of that seen in 2004. 'TCP Error' includes absence of HTTP 200 OK message and packets received greater in size than than the negotiated MSS. This was also one quarter of that seen in 2004. 'Extra retransmits' occurs when the dropped packets are retransmitted extra times. 'Not enough packets' occurs when a FIN packet is received before the end of the test. This was only 34% of what it was in 2004. '15 drop after 12 RT' occurred when packet 15 was dropped after packet 12 was retransmitted. '12 drop after 15 drop' occurred when the drop of packet 12 occurred after the drop of packet 15. These two both occurred at a low level.

The classified but ignored category: 'net unwanted drops etc.' was only 19% of the unwanted drops measured in 2004. This category also included 'Packets not sequential' and 'Packet not seen before'. The former means that the received out of order packet does not fit the existing gap, and the later indicates that an out of order packet did not have an expected sequence number.

The proportion of 'Classified servers' is more than double that in 2004, as shown in Table 5.1. 'New Reno' is more than double the previous level. 'Reno' is similar to what it was. Tahoe is just over double. 'Tahoe no fast retransmit' is reduced to 14% of the previous level. 'Aggressive Tahoe no FR' is still very low. 'Aggr FR' is still very low and half of what it was.

5.6 Conclusions

The significant contributors to the previously published errors in 2004 at 53% are 'Not enough packets' at 27% and 'HTTP error' at 16%. Now we see far fewer websites with very small amounts of data, and our use of a web crawler immediately prior to testing to gather address data prevents the use of stale addresses. 'Extra retransmits' a new error category, was not an important contributor to the error rate. Similarly out of order drops and retransmissions were not important either.

The low level of 'net unwanted drops etc' shows that the process of retrying after certain errors, mostly errors related to reordering (Unwanted drops, Packets not sequential, Packet not seen before), was successful at achieving a reduced error rate.

The reduced error rate appears to have sustained Tahoe and Reno proportions, as it is expected that they should gradually decrease. In fact, as percentages of classified servers: Reno has decreased from 15% to 10%, Tahoe no FR from 4.2% to 0.2% and New Reno has increased from 75% to 85%. The increase in New Reno is consistent with modern TCP congestion control algorithms such as BIC and Cubic being conformant with it. From these results we would expect on average slightly improved performance under congestion of webservers not implementing SACK successfully or with SACK turned off.

Chapter 6

Selective Acknowledgment

6.1 Introduction

Another development for dealing with multiple packet losses is SACK [34] [15]. The problem SACK addresses is that TCP performance may be impaired when more than one packet is lost in a data stream. This leads to the situation where it requires one RTT to learn about each lost packet, and may also lead to received packets being retransmitted. SACK involves sending of information from the receiver to the sender in acknowledgements, which indicates which blocks of packets have been successfully received. In so doing, the receiver infers which packets have been lost from it.

6.2 How SACK works

Firstly permission to operate SACK is granted during connection phase. The 'SACK permitted' option may only be sent in SYN and SYN ACK packets. When it is sent both ways SACK is established. The 'SACK' option is sent by the TCP receiver, and contains pairs of numbers that indicate edges of

sequence space that relate to data received. The bytes above and below the block between these edges have not been received. This allows the identity of multiple packets requiring retransmission to be established from one acknowledgement packet.

6.3 Related work

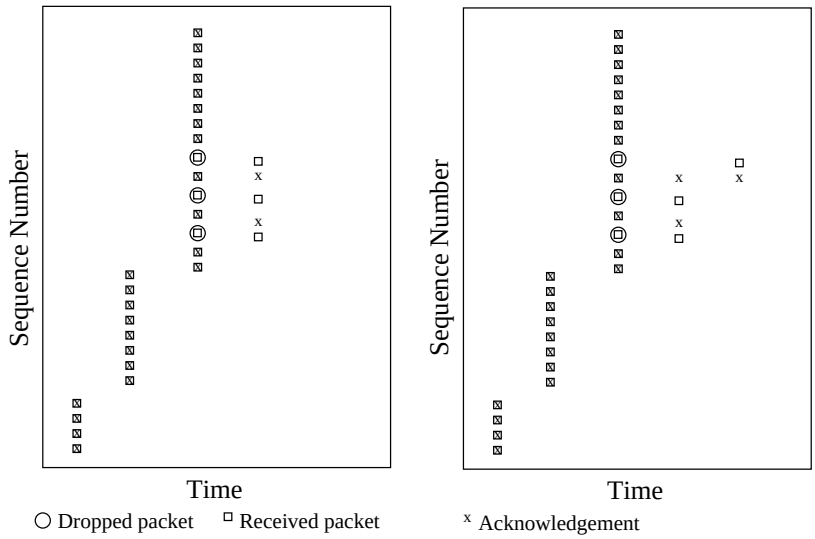
6.3.1 Sender SACK test

The TBIT sender SACK test is described in [39], where sender or receiver means that the measurement host sends or receives SACK blocks in the test respectively. The test begins by negotiating a SACK connection by sending a SYN packet with a 'SACK permitted' option. If the SYN/ACK packet does not contain a 'SACK permitted' option then the test does not continue. The testing program then drops packets 15, 17 and 19. SACKs are sent as expected, this continues until the retransmission of the dropped packets has occurred. The retransmissions of the dropped packets is timed, and interpreted after the connection is closed.

'Proper SACK' is detected when all 3 retransmissions occur in one RTT, with no unnecessary retransmissions, as shown in Figure 6.1. In 'Semi SACK' some, but not complete, use is made of SACK information. Two retransmissions occur in one RTT of dropping, but the other retransmission takes longer. In 'NewReno' no use is made of SACK information, and only responses to cumulative ACKs are observed, as shown in Figure 6.2. In 'Tahoe without fast retransmit' after an RTO, packet 15 is retransmitted, then one RTT later packets 17 and 18 are retransmitted.

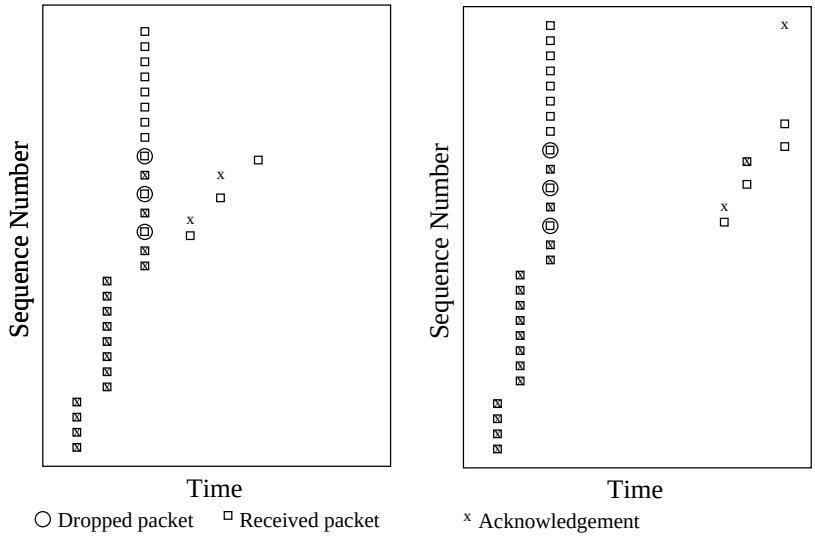
6.3.2 Receiver SACK test

The TBIT reciever SACK test is detailed in [36]. Negotiation is carried out in the same way as for sender SACK, and then an HTTP GET is split into a number of packets each containing one byte of data, the first and third packets are sent with appropriately set sequence numbers, and an ACK for each packet is in turn awaited. The second of these ACK packets should contain a SACK block which is tested for correctness. The pattern of received packets is also checked for correctness, by confirming that valid ACKs are received in response to each of the two data packets.



(a) Proper SACK. (b) Semi SACK.

Figure 6.1: Figures (a) and (b) show traces of 'proper SACK' and 'semi SACK' webservers. 'Proper SACK' retransmits all the dropped packets promptly making full use of the SACK information sent in the ACK packets. 'Semi SACK' is similar but delays retransmission of one of the packets, making only partial use of the SACK information provided.



(a) New Reno. (b) Tahoe no FR.

Figure 6.2: Figures (a) and (b) show traces of 'New Reno' and 'Tahoe no fast retransmit' webservers. 'New Reno' makes no use of the SACK information sent in the ACK packets, but does not timeout or retransmit packets unnecessarily. 'Tahoe no FR' also makes no use of SACK information but times out and resends some packets unnecessarily.

6.4 Method

The methods used were similar to TBIT SACK, however drivers were written to repeat testing when necessary. In previous research carried out in 2004 [36], for each webserver, each sender or receiver SACK test was carried out once.

6.4.1 Sender SACK test

SACK holes are the gaps between contiguous regions (blocks) that the SACK option indicates have been received. A maximum of four SACK blocks can

be sent in the options of one ACK packet. A prerun of 100 websites showed that there was 32% of cases where there were 'too many holes'. This is likely to result from gaps caused by extra dropped packets or reordering, resulting in more gaps in the byte stream received than can be represented in one ACK packet. To solve this problem, a driver was written which retried the cases where there were too many SACK holes, up to three times. Retrying is useful because the extra drop is unlikely to occur a second time and neither are the accompanying extra holes. This resulted in a net 'too many holes' error rate of 5%, a very useful improvement.

6.4.2 Receiver SACK test

In the receiver SACK test the driver had the role of carrying out a null test if the initial test failed to negotiate SACK.

6.5 Data collection

Data collection was based on active measurement of the 10000 most popular websites from Alexa. The sender test was run and warts data was collected on 20/11/2010. Similarly the receiver SACK test was run on 16/02/2011.

The data input set used for the sender and receiver IPv4/IPv6 SACK tests is the same data set as was used for IPv6 ECN. This was 1460 addresses of each IPv4 and IPv6 collected across the entire one million Alexa most popular websites. This was the same list of webservers for each type of IP address.

OS	SACK behaviour
FreeBSD 8	Semi SACK
OpenBSD 4.6	Proper SACK
NetBSD 5.0.2	New Reno (SACK permitted, non SACK)
Debian Sqeeze 2.6.32	Proper SACK
Windows XP SP2	Semi SACK

Table 6.1: Type of sender SACK behaviour observed for each operating system in the controlled environment. Two machines showed proper SACK, two showed SemiSACK and one showed non SACK behaviour.

6.6 Results

6.6.1 Sender SACK test

The machines in the controlled environment were tested with the SACK test, and the results are shown in Table 6.1. Debian Linux and OpenBSD made full use of the SACK information provided by the receiver, FreeBSD and Windows XP made partial use of SACK and NetBSD made no use at all, although all of them negotiated 'SACK permitted'.

The result for 'Proper SACK' is 45% as shown in Table 6.2. This is an increase from 18% in 2004 [36], which was measured using TBIT. The webservers which are 'not SACK capable' failed to negotiate SACK in the handshake. 'Uses SACK' is the sum of 'Proper SACK' and 'Semi SACK'. 'Does not use SACK' is the sum of 'New Reno' and 'Tahoe no FR'. 'No TCP connection' and 'Early reset' indicate a failed TCP connection. 'No data response' indicates that HTTP 200 OK was not received. 'Delayed retransmit' indicates that the time between dropping the packets and their retransmission was longer than 2 * RTT. 'Not enough packets' means that a FIN was received before the test was completed. 'No more data' means that

Result	2004	Count	Percent
Not SACK capable	29%	1790	16.9%
SACK-capable	*68%*	-	*55.2%*
Uses SACK	*27%*	-	*52.2%*
Proper SACK	18%	4737	44.7%
Semi SACK	9%	800	7.5%
Does not Use SACK	*3%*	-	*3.0%*
New Reno	2%	283	2.7%
Tahoe no FR	1%	30	0.3%
No TCP connection	2%	69	0.7%
Early reset		13	0.1%
No data response	0.5%	42	0.4%
Delayed retransmit		23	0.2%
Not enough packets	25%	757	7.1%
No more data		305	2.9%
MSS error	0%	371	3.5%
More than max packets/reordered		411	3.9%
net too many holes		970	9.2%
total IPs		10601	-
Too many holes		7831	-
Retries after too many holes		6861	-

Table 6.2: Results category counts and percentages for the sender SACK test carried out on 20/11/2010. TBIT results from 2004 [36] are also shown. The counts adding up to 'total IPs' are shown above that line and the number with 'Too many holes' and the number of retries are shown below that line. If 'Too many holes' occurs then the test is rerun up to a total of three times.

the connection timed out waiting for more data three times. 'MSS error' indicates that the negotiated packet size was exceeded by the webserver. 'More than max packets/reordered' is also contributed to by excessive reordering. 'Net too many holes' is the number of webservers classified with too many SACK holes, after retries have produced more results in other categories from webservers originally in this category. 'Total IPs' is the number of distinct webservers tested. 'Too many holes' is the total number of webservers in this category including all retries. 'Retries after too many holes' is the total number of retry tests carried out.

Most of the errors in the 2004 test were 'Not enough packets' at 25% compared to 7% here. Our test experienced greater error rates from packet reordering and drops as seen in the 'Too many holes' and 'More than max packets/reordered' results. This may be a result of using a congested vantage point location to carry out the tests or possibly a result of the existence of multiple paths.

Though the error rates are quite low, it may be possible to reduce them further by selecting the data set differently or by relaxing some constraints. 'Not enough packets' error rate could be reduced by using data size information from the web crawler to exclude webservers with data below 25 packets. The MSS packet size limit could be relaxed to reduce this error. 'More than max packets/reordered' could be included in the cases which are retried. The number of retries could be increased from 3 to 4. It is however of interest to compare some of these error rates with the 2004 results, and to compare the SACK results on a similar basis of measurement. Furthermore it may not be desirable to make selections on the data set, as this may introduce bias.

'Semi SACK' and 'New Reno' had similar values to 2004, however 'Semi SACK' is now much less than 50% of the 'Uses SACK' group i.e. 14%. This indicates that most operating systems deployed using SACK information now implement SACK correctly. 'Tahoe no FR' has reduced to about 30% of what it was. This might be expected as Tahoe is now associated with obsolete operating systems. Some of the classified cases in these categories other than 'Proper SACK' could be affected by middleboxes, as SACK blocks may not be correctly translated.

Analysis of HTTP server specification within the SACK results categories was carried out. Of the 'Proper SACK' machines, 23% were positively identified as Linux, however the categories 'Apache' and 'nginx' made up most of the rest, with Microsoft-IIS at 4% and no server identifier at 11%. Microsoft-IIS is well represented in the 'Semi SACK' category at 40% of webservers. 5% of these webservers were identified as running a Linux operating system. In the 'New Reno' category 7% were positively identified as Linux, and 13% Windows. In the 'Tahoe no FR' category no Linux machines were positively identified, and 13% Windows was found. It should be noted that categories 'Apache', 'nginx' and 'Unspecified' together make up more than 50% of servers. This makes interpretation difficult.

The results of sender SACK via IPv4 and IPv6 are shown in Table 6.3. It is likely that access via both types of IP to the same operating system will produce similar results. 'Proper SACK', 'Semi SACK' and 'Tahoe' are higher in IPv4, whereas 'New Reno' is similar. The higher levels of 'No TCP connection' and 'Early reset' may explain these differences. Other than these differences the errors appear similar with eachother.

Result	Count 4	Percent 4	Count 6	Percent 6
Not SACK capable	67	4.5%	83	5.6%
SACK-capable	-	*71.6%*	-	*60.0%*
Uses SACK	-	*70.4%*	-	*59.1%*
Proper SACK	997	67.4%	865	58.2%
Semi SACK	45	3.0%	14	0.9%
Does not Use SACK	-	*1.2%*	-	*0.9%*
New Reno	10	0.7%	13	0.9%
Tahoe no FR	7	0.5%	0	0%
No TCP connection	14	0.9%	186	12.5%
Early reset	7	0.5%	38	2.6%
No data response	1	0.1%	0	0%
Delayed retransmit	0	0%	2	0.1%
Not enough packets	265	17.9%	238	16.0%
No more data	23	1.6%	18	1.2%
MSS error	39	2.6%	21	1.4%
More than max packets/reordered	1	0.1%	4	0.3%
net too many holes	4	0.3%	3	0.2%
total IPs	1480	-	1485	-
Too many holes	32	-	30	-
Retries after too many holes	28	-	27	-

Table 6.3: Results category counts and percentages for the IPv4 and IPv6 sender SACK test carried out on 7/2/2011. The Count and Percent column headings have a suffix which specifies the type of IP. The counts adding up to 'total IPs' are shown above the bottom line and the number with 'Too many holes' and the number of retries are shown below that line. If 'Too many holes' occurs then the test is rerun up to a total of three times.

OS	SACK behaviour
FreeBSD 8	Success
OpenBSD 4.6	Success
NetBSD 5.0.2	Success
Debian Sqeeze 2.6.32	Success
Windows XP SP2	Success

Table 6.4: Type of receiver SACK behaviour observed for each operating system in the controlled environment. All of the webservers exhibited successful sending of SACK blocks and interaction with the SACK test.

The IPv4 only results give a lower result for 'Proper SACK' and a higher result for 'Semi SACK' and 'New Reno'. Apart from 'Not enough packets' which is lower in IPv4 only, some of the errors are higher, as is 'Not SACK capable'. This is consistent with the machines addressable by IPv6 having more modern operating systems and a greater likelihood of being SACK capable.

6.6.2 Receiver SACK test

The receiver SACK test was executed in the controlled environment and the results are shown in Table 6.4. This shows that all of the systems tested were able to send valid SACK blocks.

The proportion of 'SACK blocks OK' is 81% as shown in Table 6.5. This is an increase from 65% in 2004 [36]. The 'Not SACK capable' level has reduced by 40% and the error rate mostly made up of 'No connection' cases has reduced to a low level. Small numbers of 'Shifted SACK blocks' and null 'Success' are both indicators of possible middlebox interference with SACK.

Result	2004	Count	Percent
Not SACK capable	28.8%	1747	16.6%
SACK blocks OK	64.7%	8511	81.1%
Shifted SACK blocks	0.5%	82	0.8%
Packet sequence error	0.1%	54	0.5%
No block	0.1%	56	0.5%
No connection	5.3%	42	0.4%
Early reset	0.4%	3	0.0%
total IPs		10495	-
Null			
Success	-	6	0.1%
No connection	-	34	0.3%
Early reset	-	5	0.1%

Table 6.5: Results category counts and percentages for the receiver SACK test carried out on 16/02/2011. TBIT results from 2004 [36] are also shown. The null test referred to in the ECN chapter is run if the test fails to negotiate a SACK capable connection.

The results for IPv4/IPv6 receiver SACK testing are shown in Table 6.6. 'SACK blocks OK' for IPv6 is lower than for IPv4 and this seems to be largely explained by increases in 'Packet sequence error' and 'No connection'. The level of 'Not SACK capable' seems to be similar for the two types of IP address. 'Shifted SACK blocks' and null 'Success' levels indicate only a very low level of possible middlebox interference.

6.7 Conclusions

6.7.1 Sender SACK test

There has been a decrease in errors from 40% in 2004 [36] to 28% in this research. There is also a change in composition as 'Not enough packets' and

Result	Count 4	Percent 4	Count 6	Percent 6
Not SACK capable	71	4.8%	88	5.9%
SACK blocks OK	1371	92.6%	881	59.3%
Shifted SACK blocks	3	0.2%	3	0.2%
Packet sequence error	3	0.2%	245	16.5%
No block	10	0.7%	45	3.0%
No connection	21	1.4%	177	11.9%
Early reset	1	0.1%	46	3.1%
total IPs	1480	-	1485	-
Null				
Success	0	0.0%	2	0.1%
No connection	20	1.4%	174	11.7%
Early reset	2	0.1%	46	3.1%
No data	0	0.0%	1	0.1%
total	22	-	223	-

Table 6.6: Results category counts and percentages for the IPv4 and IPv6 receiver SACK test carried out on 15/2/2011. The Count and Percent column headings have a suffix which specifies the type of IP. A null test is run if the test fails to negotiate a SACK capable connection.

'HTTP error' are reduced in proportion and 'too many holes' is increased. This may be a result of there being less stale URLs but a more congested Internet access point currently. The frequency of 'too many holes' would have been much greater if not for the use of the driver. This error was built in to the original TBIT program but was not reported in the 2004 results.

Modern successful SACK implementations in Linux make up the great proportion of 'Proper SACK' capable webservers, and though some Windows systems are SACK successful the proportion is still quite low.

Globally routable IPv6 addressable machines are more likely to implement proper SACK than IPv4 only, and on the same machine SACK capable IPv4 is more likely to fail to implement proper sack. This latter situation may however be associated with cases where IPv6 fails to connect.

6.7.2 Receiver SACK test

The high success rate in the controlled environment suggests that most webservers are capable of sending SACK blocks. One might therefore expect that many non success cases seen might be related to middlebox interference.

Successful SACK for popular webservers with IPv4 addresses has increased at the expense of errors and SACK incapable cases. Strangely the IPv4 success rate for webservers with IPv6 addresses is higher than for IPv4 only, whereas the IPv6 success rate itself is lower than this level, though failed IPv6 connections and failed packet sequences are major contributors to this latter situation.

6.7.3 Overall

More than half of high traffic webservers use SACK information, and most of these make full and proper use of this, and for sender SACK the rate of Semi-SACK has decreased. These are large increases in the correct implementation of SACK, a benefit to the Internet.

A small amount of middlebox interference was detected. This is likely to mean that SACK blocks are being incorrectly recalculated, or remaining unchanged when conversion is necessary along with transformed sequence and related numbers. Interference could also involve blocked SACK connections, where the 'SACK permitted' option is interfered with. The amount of SACK interference appears to be insufficient to hinder its uptake and activation.

Chapter 7

Initial congestion window

7.1 Introduction

ICW is the value of the congestion window that occurs at the beginning of a connection. The value of this parameter affects the speed at which the initial slow start accelerates, and thus affects TCP performance. This setting has been changed to allow smaller packets sizes to occur in greater numbers at connection start. It is thus of interest to know if this specification change has been adopted and what impact on TCP performance has occurred. In this chapter details of ICW algorithms are explained and measurements are carried out on the prevalence of different values.

7.2 Initial congestion window

The ICW for slow start when a connection is initialised was originally set at less than or equal to 2*MSS and not more than 2 segments [4]. This has recently been relaxed [1] [2] to: 4*MSS for MSS <= 1095, 3*MSS for MSS <= 2190 and 2*MSS for MSS > 2190. These changes were made for

a number of reasons. A disadvantage of starting with one segment is that a receiver implementing delayed acknowledgements may timeout before continuing. Connections carrying out small data transfers will be quicker than previously. This improves high bandwidth high delay connections by removing up to three RTT delays and one RTO delay [1].

7.3 Related work (TBIT)

Firstly TBIT establishes a TCP connection and an MSS option of 256 bytes is specified, after this an HTTP request is made for the web page. None of the resulting data packets are acknowledged, so eventually the first data packet is resent by the server after a congestion window full of packets has been sent. The data packets are counted to give initial congestion window. If a FIN packet is received from the server then the test fails, as the size of the window is likely to have been limited by lack of data.

7.4 Method

The TBIT test was adapted into a scamper module. Scamper's state struct was extended to cover necessary TBIT variables, and the scamper routine to process a received packet was connected to the TBIT function for this purpose. A scamper function to send a data packet was also called at the appropriate time. The main deviation from the TBIT regime, where the test was run once for each website in the later analysis [36], was that a driver was used to run the test twice, once at each of two different MSS settings.

7.5 Data collection

The top 10000 servers of the most popular and thus most commonly frequented websites were considered to be a useful and representative sample of Internet traffic providing hosts. The Alexa top one million websites were downloaded. A perl web crawler program was run to gather the 10000 most popular websites and their IP addresses in a correctly formatted data file. This data file was assembled on the 29/10/2010 and the test was run using the driver set on MSS 256 and 1460 bytes. The ICW test produced a warts data file and this was analysed after the test was finished.

7.6 Results

Once the ICW test was initially assembled, it was run in a controlled environment where several machines running various operating systems were connected to a test machine via a 200ms delay. The test machine was used to run the ICW TBIT test on these isolated servers. Table 7.1 shows the observed behaviour of the common operating systems tested, where the expected behaviour is shown by one operating system, which is Debian Sqeeze Linux.

The total errors came to a similar value for each of the MSS settings, which was a fairly low 17%, as shown in Table 7.2. The breakdown of these is as follows. 'No TCP connection' occurred when no SYN/ACK was returned after an initial SYN was sent, and the SYN was sent a further two times. 'Early reset' was when a RST packet was received from the web server at connection time.

OS	256 MSS	1460 MSS
FreeBSD 8	1	1
OpenBSD 4.6	5	4
NetBSD 5.0.2	4	4
Debian Sqeeze 2.6.32	4	3
Windows XP SP2	2	2

Table 7.1: Initial congestion window behaviour observed for each operating system in the controlled environment. The numeric result is the number of packets making up the initial congestion window.

Error type	256	MSS	1460	MSS
No TCP connection	118	1.1%	105	1.0%
Early reset	81	0.8%	62	0.6%
No data response	119	1.1%	114	1.1%
TCP Error	328	3.1%	41	0.4%
HTTP error	216	2.1%	205	2.0%
Not enough packets	101	1.0%	553	5.3%
MSS error	383	3.7%	0	0.0%
Length error	0	0.0%	374	3.6%
Too many packets	103	1.0%	26	0.2%
Early reordering	315	3.0%	310	3.0%
Total errors	1764	17.2%	1790	17.2%

Table 7.2: These are the errors from an ICW test of 10424 websites on the 29/10/2010. There is a count of errors and a percent value for each of MSS 256 and 1460.

'No data response' was when no HTTP data was received after the HTTP request was sent, this is similar to a count of window size zero but may also contain cases where another error occurred, though most of these will have been an HTTP error. 'TCP error' was when the initial window estimate was greater than the number of packets received, this is mostly cases of dropped packets. 'HTTP error' occurred when an HTTP response code of 200 OK was not received in the first data packet.

'Not enough packets' occurred when a FIN packet was received from the server, indicating that there was no more data and the full congestion window may not have been seen.

'MSS error' occurred when the segment size specified by the MSS option sent in the SYN packet was exceeded, this occurred in 4% of cases for MSS 256 but not at all for 1460. This is likely to be because the common Maximum Transmission Unit (MTU) for ethernet is 1500, thus the segment size is usually 1460, without the TCP and IP headers, and is commonly not exceeded. On the other hand MSS 256 is fairly small and in some cases may be exceeded by operating systems that ignore the MSS option requirement. Initially 'Length error' occurred when the packets where shorter than that specified by the MSS setting of the test. After the test was run it was decided to reclassify these as valid if the data packets were all larger or all smaller than the transition value of 1095 segment size, and in line with the MSS setting of the test. This reduced the MSS 1460 error measurement from 15% to 4% and the MSS 256 measurement from 3% to 0%. The now valid packet streams were counted in the warts data file to give the additional ICW counts.

'Too many packets' was when the measured window size was larger than 29 packets, and these same numbers are used for the 'more' category of the results table. MSS 256 showed a larger rate of occurrence of this error, this may have something to do with the pattern of allowing greater numbers of smaller packets, though greater than 29 packets is really a somewhat extreme case of this.

'Early reordering' occurred when the sequence number of the first data packet received was greater than that of a subsequent data packet. This requirement was later relaxed to allow out of order data packets at the beginning of the trace.

For each MSS setting, the second largest count is the expected result [2] 4 or 3 for MSS 256 and 1460 respectively, and the largest is 2 packets, as shown in Table 7.3. Looking at this result in terms of the results from the controlled environment and knowing that windows is a popular operating system, one might deduce that windows is producing a predominance of 2 results, assuming that other windows operating systems behave in the same way as XP, and Linux is responsible for producing the second largest counts correctly. It can also be noted MSS 256 has a larger count above ICW of 6 than MSS 1460.

Results published by others from 2004 [36] reported that 2% of servers had results of 3 or 4 segments and 1% were larger. These results were reported for testing based on an MSS of 256. We have seen a change in the former 3 or 4 segments group, increasing to about 20% for the appropriate MSS, and the later group (ICW $>$ 4 segments) at MSS 256 is largely unchanged.

init cwnd	256	MSS	1460	MSS
1	120	1.2%	333	3.2%
2	6200	59.5%	6177	59.3%
3	264	2.5%	1359	13.0%
4	1595	15.3%	581	5.6%
5	303	2.9%	39	0.4%
6	38	0.4%	73	0.7%
7	29	0.3%	9	0.1%
8	17	0.2%	9	0.1%
9	16	0.2%	5	0.0%
10	27	0.3%	7	0.1%
11	8	0.1%	3	0.0%
12	7	0.1%	3	0.0%
13	10	0.1%	4	0.0%
14	3	0.0%	2	0.0%
15	4	0.0%	0	0.0%
16	3	0.0%	2	0.0%
17	1	0.0%	2	0.0%
18	7	0.1%	0	0.0%
19	1	0.0%	2	0.0%
20	1	0.0%	1	0.0%
22	1	0.0%	3	0.0%
23	2	0.0%	7	0.1%
26	2	0.0%	1	0.0%
29	1	0.0%	1	0.0%
more	103	1.0%	26	0.2%
>6	243	-	87	-

Table 7.3: These are the results from an ICW test of 10424 websites on the 29/10/2010. There is a count of websites with a given initial congestion window size and a percent value, for each of MSS 256 and 1460.

init cwnd	256 MSS	1460 MSS
Microsoft-IIS/6.0	12	24
Microsoft-IIS/7.0	3	7
Apache-Coyote/1.1	3	6
Apache/2.2.3 (CentOS)	36	1
LiteSpeed	5	0
nginx	8	0
Apache/2.2.3 (Red Hat)	17	3
Squeegit	1	0
Apache/2.2.4 (Fedora)	2	0
Oracle Application Server	1	1
IBM_HTTP_Server	10	3
Sun-ONE-Web-Server/6.1	5	1
Zeus/4_3	5	1
PWS/1.6.2	1	0
Resin/3.1.8	0	1
Mongrel	2	0
Unix	14	8
Apache	83	16
no value	35	15
total	243	87

Table 7.4: These are the prolific servers from an ICW test of 10424 websites on the 29/10/2010. There is a count of servers producing more than 6 ICW packets, for each of MSS 256 and 1460.

Many servers specified Apache or no value at all, however there are enough servers specified to give an indication of the population. Table 7.4 shows that from MSS 256 to 1460 Red Hat and CentOS show a decrease and Microsoft-IIS/6.0 shows an increase. The Microsoft trend is particularly pronounced with Microsoft-IIS/6.0 making up about 30% of the MSS 1460 prolific group. CentOS is next in frequency making up 15% of the MSS 256 prolific group.

7.7 Conclusion

'TCP error', 'HTTP error' and 'early reordering' are the main contributors to errors in the MSS 256 group. For the MSS 1460 group 'Not enough packets' shows up as an important error contribution also. Both sets of errors are low at 17%. The effects of congestion and perhaps multiple paths seem apparent, and for the larger MSS group, lack of data in some cases.

These results suggest that about 75% of classified web servers adhere to the older specification of ICW, some 20% have changed to the newer standard, and some small 3% are prolific or have a large ICW. Some specific operating systems may be of interest for further study of why some prolific ICW traces occur, however better identification of specific operating system versions is needed to do this.

Chapter 8

Conclusions

The implementation rates of a number of TCP congestion control related algorithms have been measured. Window reduction due to loss has shown a large transition to BIC or Cubic style behaviour since 2004. ECN has shown a steady increase of successful implementation from a low level over several months. This is a 16 fold increase since 2004. Interference by middle boxes seems to have reduced to a level where ECN is not being switched off to the extent that it was. ECN success has shown to be twice as likely in machines that have interfaces with globally routable IPv6 addresses. These machines are more likely to have more up to date ECN capable and ECN activated operating systems. It is possible that the higher rate of 'no ECN echo' for IPv4 is due to middleboxes which block IPv4 ECN echos and not IPv6. In reaction to loss, Tahoe has remained steady as a percentage of classified servers, Reno has reduced and New Reno has greatly increased. This is consistent with new TCP protocols adopting New Reno conformant behaviour. Nearly half of webservers successfully implement SACK as measured by the sender SACK test. Proper SACK implementation has increased by 250% since 2004. Implementation of proper SACK in IPv6 addressable

webservers was higher than for IPv4 only. Though proper SACK was more frequent for IPv4 than IPv6 for the same set of machines, semi SACK was also more frequent. As for ECN the IPv6 addressable machines seem more likely to have more modern proper SACK capable operating systems. Another possible cause of higher non proper SACK, SACK capable results, as seen for IPv4 is middleboxes. This is because middleboxes sometimes fail to properly translate SACK blocks. In the receiver SACK test case, high success rates are seen, and increases are seen from previously measured levels. We also see some cases of shifted SACK blocks which is an indicator of possible middlebox interference, at a low level. When measuring over IPv6 webservers, success tends to be higher unless there are significant error rates. ICW protocols have transitioned to 20% implementation of the new standard of 3 or 4 packets rather than 2 packets.

There is a general trend of operating system designers and system administrators to conform with these RFC directions, in the area of congestion control, if slowly at times. This is good for the Internet because the performance and stability experienced by the end user is improving. This situation is also better for ISPs and other providers of Internet services.

Further work could involve monitoring TCP status on the Internet with repeats of the same tests or analysis with tests upgraded to cater for change. In particular this could include testing IPv6 webservers with an array of tests. It would also be recommended to follow up the anomalies found in this research, such as inappropriate CWR packets, shifted SACK blocks, larger than expected ICWs, and to study the relationship of congestion window reduction on loss to congestion measures. It would also be helpful to see an increase

in the detail about operating systems available in the server specification provided by HTTP. This could lead to more detailed analysis of particular operating system involvement in specific diagnosed TCP behaviours.

Bibliography

[1] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. *RFC3390*, 2002.

[2] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. *RFC5681*, 2009.

[3] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. *RFC2581*, 1999.

[4] Mark Allman, Vern Paxson, and W. Richard Stevens. TCP congestion control. *RFC 2581*, 1999.

[5] M. Arlitt and C. Williamson. An analysis of TCP reset behaviour on the Internet. *ACM SIGCOMM Computer Communication Review*, 35:37–44, 2005.

[6] J. Bellardo and S. Savage. Measuring packet reordering. *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 97–105, 2002.

[7] R. Beverly. *Passive and Active Network Measurement*, chapter A Robust Classifier for Passive TCP/IP Fingerprinting, pages 158–167. Springer-Verlag, 2004.

[8] R. Braden. Requirements for Internet Hosts - Communication Layers. *RFC1122*, 1989.

[9] R. Braden, D. Borman, and C. Partridge. Computing the Internet checksum. *RFC1071*, 1988.

[10] M. P. Brig. Projected Impacts of the Internet Protocol version 6 (IPv6) on the USN and USMC Enterprise. *SPAWAR Systems Center Charleston www.cav6tf.org/articles/ipv6impactreport.doc*, pages 1–25, 2002.

[11] D.D. Clark. Window and acknowledgement strategy in TCP. *RFC813*, 1982.

[12] D.Antoniades, M.Athanatos, A.Papadogiannakis, E.P.Markatos, and C. Dovrolis. Available bandwidth measurement as simple as running wget. *Proceedings of Passive and Active Measurements (PAM)*, 2006.

[13] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. *RFC2460*, 1998.

[14] S.J. Eichler. Quantifying ECN implementation in the Internet. *Comp514 report*, 2009.

[15] K. Fall and S. Floyd. Comparisons of Tahoe, Reno, and Sack TCP. *Lawrence Berkeley National Laboratory*, pages 1–14, 1995.

[16] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM SIGCOMM Computer Communication Review*, 26(3):5–21, 1996.

[17] S. Feyzabadi and J. Schonwalder. Identifying TCP congestion control algorithms using active probing. *PAM2010 conference*, page 1, 2010.

[18] S.S. Feyzabadi. Identifying TCP congestion control mechanisms using active probing. *cnds.eecs.jacobs-university.de/courses/nds-2009/feyzabadi-report.pdf*, 2009.

[19] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7:458–472, 1999.

[20] S. Floyd and T. Henderson. The NewReno Modification to TCPs Fast Recovery Algorithm. *RFC2582*, 1999.

[21] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. *RFC3782*, pages 1–19, 2004.

[22] Sally Floyd. Congestion control principles, 1999.

[23] Rodrigo Fonseca, George Manning Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. IP Options are not an option. Technical Report UCB/EECS-2005-24, EECS Department, University of California, Berkeley, Dec 2005.

[24] ITU. Information technology - Open systems interconnection - Basic reference model: The basic model. *ITU-T Recommendation X.200*, 1994.

[25] V. Jacobson. Congestion avoidance and control. *Proceedings of SIGCOMM 88*, pages 1–25, 1988.

[26] V. Jacobson. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, 25:157 – 187, 1995.

[27] Elliott Karpilovsky, Alexandre Gerber, Dan Pei, Jennifer Rexford, and Aman Shaikh. Quantifying the Extent of IPv6 Deployment. In Sue Moon, Renata Teixeira, and Steve Uhlig, editors, *Passive and Active Network Measurement*, volume 5448 of *Lecture Notes in Computer Science*, pages 13–22. Springer Berlin / Heidelberg, 2009.

[28] A. Kuzmanovic, A. Mondal, S. Floyd, and K. Ramakrishnan. Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets. *RFC5562*, 2009.

[29] S. Ladha, P.D. Amer, A.Caro, and J.R. Iyengar. On the prevalence and evaluation of recent TCP enhancements. *IEEE Globecom*, 3:1301–1307, 2004.

[30] A. Langley. Probing the viability of TCP extensions. Technical report, Google Inc., 2009.

[31] M. Luckie, K. Cho, and B. Owens. Inferring and debugging path MTU discovery failures. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005.

[32] M. Luckie and B. Stasiewicz. Measuring path MTU discovery behaviour. *IMC'10*, 2010.

[33] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure, USA, 2009.

[34] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. *RFC2018*, 1996.

[35] A. Medina, M. Allman, and S. Floyd. Measuring interactions between transport protocols and middleboxes. *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 336–341, 2004.

[36] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the Internet. *ACM SIGCOMM Computer Communication Review*, 35:37–52, 2005.

[37] B. Moraru, F. Copaciu, G. Lazar, and V. Dobrota. Practical analysis of TCP implementations: Tahoe, Reno NewReno. *RoEduNet International Conference*, pages 125–130, 2003.

[38] J. Nagle. Congestion control in IP/TCP Internetworks. *ACM SIGCOMM Computer Communication Review*, 14, 1984.

[39] J. Padhye and S. Floyd. Identifying the TCP behavior of web servers. *ACM SIGCOMM*, pages 1–13, 2000.

[40] J. Padhye and S. Floyd. On inferring TCP behavior. *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 287–298, 2001.

[41] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP throughput: a simple model and its empirical validation. *Proceedings of SIGCOMM*, pages 303–314, 1998.

[42] V. Paxson. Automated packet trace analysis of TCP implementations. *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 167–179, 1997.

[43] V. Paxson and M. Allman. Computing TCP's retransmission timer. *RFC2988*, 2000.

[44] Jon Postel. Transmission control protocol. *RFC793*, 1981.

[45] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN). *RFC3168*, pages 1–63, 2001.

[46] I. Rhee and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[47] J.P. Robinson, M. Kestnbaum, A. Neustadtl, and A. Alvarez. Mass Media Use and Social Life Among Internet Users. *Social Science Computer Review*, 18:490–501, 2000.

[48] M. T. Rose and D. E. Cass. OSI transport services on top of the TCP. *Computer Networks and ISDN Systems*, 12:159–173, 1986.

[49] N.K.G. Samaraweera. Non-congestion packet loss detection for TCP error recovery using wireless links. *Communications, IEE Proceedings*, 146:222–230, 1999.

[50] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: informed Internet routing and transport. *Micro, IEEE*, 19:50–59, 1999.

[51] S. Shakkottai, N. Brownlee, and K.C. Claffy. A Study of Burstiness in TCP Flows. *Passive and Active Network Measurement, Lecture Notes in Computer Science*, 3431:13–26, 2005.

[52] S. Shakkottai, R. Srikant, N. Brownlee, A. Broido, and K.C. Claffy. The RTT distribution of TCP flows in the Internet and its impact on TCP-based flow control. 2004.

[53] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit. *RFC2001*, 1997.

[54] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (BIC) for fast long-distance networks. *IEEE Infocom*, pages 1–11, 2004.