INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

Performance Analysis of TCP's Retransmission Timeout Mechanism

by

Nabil Seddigh
B. A. Sc., University of Waterloo

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Engineering

Ottawa-Carleton Institute of Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
December 20, 2000

Ottawa, Ontario
© 2000 Nabil Seddigh

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-61025-X

Canada

The undersigned recommend to the Faculty of Graduate Studies and Research
acceptance of the thesis

**Performance Analysis of TCP's
Retransmission Timeout Mechanism**

submitted by

Nabil Seddigh

B.A. Sc.

in partial fulfillment of the requirements for
the degree of Master of Engineering

Thesis Supervisor

Chair, Department of Systems and
Computer Engineering

Carleton University

Abstract

TCP is the dominant reliable transport protocol utilized in the Internet. Its robust sender flow control mechanism allows it to gently probe the network to determine the fair-share bandwidth for an individual flow. Much of the recent work on improving TCP flow control has attempted to reduce the possibility of timeout since timeouts cause a drastic performance reduction for the flow. Detailed studies have been performed to analyze and propose improvements for retransmission timeouts (RTO) experienced by the TCP flow after it has sent a few packets. However, to date, there has been no study to determine the effect of the initial timeout value that is currently recommended in the TCP standard.

This work undertakes an extensive investigation into TCP's initial RTO value. First, a systematic experimental analysis of packet drops for a single TCP flow is undertaken. The goal is to identify those packets whose loss would result in a timeout with the initial RTO value. Second, a large-scale study of Internet traffic traces is performed. The objective is to determine loss rates for those packets that cause timeout with the initial RTO value. This provides the motivation for this work. Third, experiments are performed using nine commercial operating systems to understand the initial RTO value utilized by TCP implementations today.

The final part of this work performs simulations to study the impact of the initial RTO values as used in TCP implementations today. The inputs to the simulation are obtained from the results of the commercial operating system study. For the study, we develop a model of HTTP traffic in a RED-based router network. We perform simulations to determine the impact of the initial RTO value on the transfer delay of web objects. The results are compared using a set of defined performance indicators. The simulations are performed using both a single long run as well as a completely randomized multiple run method.

Finally, the outcome of this work is a recommendation to modify the current initial RTO value recommended in the TCP standards.

"A mechanism of world inter-communication will be devised, embracing the whole planet, freed from national hinderances and restrictions, and functioning with marvellous swiftness and perfect regularity."

-- Shoghi Rabbani, The Unfoldment of World Civilization, 1936

Acknowledgements

The completion of my Masters' degree is largely due to the inspiration and support of a number of people.

I am indebted to my parents who have always encouraged me to strive for the highest standards of education and who, along with my siblings, provided constant gentle reminders along the road to completing this degree.

A debt of gratitude is owed to my wife May for her endearing patience and abiding support during the late nights when my thesis research was in progress.

I am grateful to Dr Mihail Devetsikiotis, my thesis supervisor who provided insightful guidance, assisted me to narrow my thesis focus and whose emphasis on simulation correctness led to the detailed simulation experiments carried out. I also acknowledge the contributions of Dr Ioannis Lambadaris and Dr Luis Orozco-Barbosa whose careful review of the thesis helped in improving its quality.

My colleague at Nortel, Biswajit Nandy, was a source of constant encouragement, and provided insightful comments on early versions of the thesis. I also recognize the much-appreciated assistance by other colleagues at Nortel on a variety of technical matters related to this thesis work. Peter Piedad and Rupinder Makkar assisted with HTTP traffic models used in the experiments. Jamal Hadi Salim helped with investigation into the TCP timeout mechanisms in Linux. Fulu Li provided some much-needed help and starter perl scripts for analysis of the Internet traffic traces.

Finally, I gratefully acknowledge the financial support of Nortel Networks towards completion of this degree.

Table of Contents

Chapter 1	Introduction	1
1.1	Thesis Background and Motivation	1
1.2	Thesis Objectives	3
1.3	Thesis Organization	4
Chapter 2	Background and Related Work	5
2.1	TCP Overview	5
2.2	TCP Flow Control.....	6
2.3	TCP's Retransmission Mechanism – Related Work.....	8
2.4	Initial RTO Value – Examination of Related Work	10
Chapter 3	When is Initial RTO Important?	12
3.1	Experimental Approach	12
3.2	Experimental Methodology	12
3.3	Observations	14
3.4	Analysis.....	15
Chapter 4	Internet Traffic Study	16
4.1	Network Topology	17
4.2	Data Set.....	18
4.3	Analysis Methodology	19
4.4	Experimental Results and Analysis	22
4.4.1	Per-Flow Statistics – Observations.....	22
4.4.2	SYN and SYNACK Loss Rates - Observations	24
4.4.3	Broken TCP Implementations	25
Chapter 5	TCP Implementation Study	28

5.1	Study of 9 Operating Systems – What Initial RTO Is Used?.....	28
5.2	Methodology	28
5.3	Experimental Results – Initial RTO Value	29
5.4	Observations And Analysis.....	29
5.5	Exponential Backoff Results.....	30
Chapter 6	Simulation Setup	32
6.1	Network Topology	32
6.2	Traffic Model – Background FTP Flows.....	34
6.3	Traffic Model – HTTP Flows	34
Chapter 7	Simulation Results	38
7.1	Performance Metrics	38
7.2	Traffic Model – HTTP Flows	40
7.3	Impact of Loss Rate	42
7.3.1	Experimental Results and Observations	42
7.3.2	Analysis	44
7.4	TCP Timer Granularity	45
7.4.1	TCP Short Flow Traffic Model	45
7.4.2	Experimental Results and Observations – Timer Granularity	46
7.4.3	Analysis – Timer Granularity	48
7.5	Impact of RTT Size.....	48
7.5.1	Borderline Large RTT experiment	48
7.5.2	Large RTT – Observations	49
7.5.3	Large RTT – Analysis	52
7.5.4	Common Internet RTT and Small TCP Tick Timer - Experiment.....	53
7.5.5	Common RTT – Observations.....	53
7.5.6	Common RTT - Analysis	54

7.6	Validation of Simulation Methodology	55
7.6.1	Single-Factor Completely Randomized Experimental Design.....	55
7.6.2	ANOVA – Approach and Experimental Setup.....	56
7.6.3	ANOVA – Results and Analysis	58
Chapter 8	Discussion	60
Chapter 9	Conclusion and Future Work	62
9.1	Conclusion	62
9.2	Future Work	63
Chapter 10	References	64

List of Figures

Figure 1	CWND vs Time for a TCP flow	7
Figure 2	Network Topology – Engineered Packet Loss for Linux TCP.....	13
Figure 3	VBNS Network Topology	16
Figure 4	VBNS sites used in Traffic Study.....	17
Figure 5	Per-Flow Packet Count Distribution.....	22
Figure 6	Per-Flow Byte Count Distribution.....	23
Figure 7	Per-Flow Retransmit-Burst (Loss) Count Distribution	24
Figure 8	SYN and SYNACK Relative Loss Rates	25
Figure 9	Lossy Flows with SYN and SYNACK Loss	26
Figure 10	Exponentially Backed Off RTO Values for 9 Operating Systems.....	31
Figure 11	Simulation Network Topology.....	33
Figure 12	Page Size Distribution.....	40
Figure 13	Inter-page think-time distribution	41
Figure 14	Inter-object time distribution.....	41
Figure 15	Initial RTO Effect for Different Loss – Average Delay.....	42
Figure 16	Initial RTO Effect for Different Loss – Delay Standard Dev	43
Figure 17	Initial RTO Effect for Different Loss Rates –Transfer Delay Fairness	44
Figure 18	Initial RTO Value Effect for Different Timer Granularities–Avg Delay.....	45
Figure 19	Initial RTO Value Effect for Different Timer Granularities – Std Dev	46
Figure 20	Initial RTO Value Effect for Different Timer Granularities – Fairness.....	47
Figure 21	Average Delay – Common RTT; Small Timer	53
Figure 22	Standard Deviation – Common RTT; Small Timer	54
Figure 23	Fairness Index – Common RTT; Small Timer.....	54

List of Tables

Table 1:	Description of 71 Data Sets.....	18
Table 2:	Average Initial RTO Values.....	29
Table 3:	Simulation Parameters	34
Table 4:	HTTP Traffic Model Parameters.....	36
Table 5:	Avg Transfer Delay for Different RTTs (s)	49
Table 6:	Standard Deviation for Different RTTs	50
Table 7:	Fairness Index for Different RTTs	51
Table 8:	Number of Timeouts – Varied RTT	52
Table 9:	Transfer Delay for 19 Repetitions of Test.....	57
Table 10:	ANOVA Analysis	58
Table 11:	Average Delay - Comparing Experimental Methods	59

Chapter 1 Introduction

One of the miracles of the twentieth century is the emergence of the Internet as a mesh of networks that connect individuals, organizations, business and government from all corners of the earth. The global reach and success of the Internet is partially due to its flexibility and evolutionary ability to cope with new heterogeneous networks, transmission media, protocols and applications. This flexibility to evolve is a testament to the robustness of the Internet's underlying TCP/IP suite of protocols.

1.1 THESIS BACKGROUND AND MOTIVATION

In particular, one of the sustaining pillars of the Internet during its remarkable growth rate over the last seven years is the approach to flow and congestion control. Historically, the prevailing model of traffic treatment in the Internet is "Best Effort" - routers treat all traffic equally and no preference or priority is accorded any data packets. Given such a model, the growth and rise of the Internet becomes even more surprising. In a Best Effort network, all end-users could perceivably act in their own best interests and forward data into the network as fast as possible in order to complete their data transfers in the shortest possible time. However, such behaviour if repeated by a critical mass of users can result in "congestion collapse" where both the end-user and the network suffer drastically reduced performance. In fact, as documented in [Ja88], in October 1986, the Internet was affected by a series of "congestion collapses" where end-users experienced data throughput rates reduced by a factor of a thousand. This "collapse" was due to the rudimentary flow control mechanisms in the dominant Internet transport protocol of the time - TCP (Transmission Control Protocol). In "congestion collapse", the majority of the traffic sent by end-hosts is discarded by routers due to congestion or full queues.

Jacobson's 1988 work [Ja88] proposed a series of enhancements to the original TCP flow control mechanism to address the "congestion collapse" issue. The enhancements use the "conservation of packets" principle to achieve network stability.

To adhere to such a principle, an end-host in ‘equilibrium’ only inserts new data packets into the network when packets in transit have exited. A TCP flow is in ‘equilibrium’ when it has a full fair-share window of packets in flight (transit) between source and destination.

Thus, the current TCP is a “network-friendly” protocol that increases its sending rate during network idle times and reduces its sending rate as the network becomes more congested. It uses packet loss as an indicator of network congestion and adjusts its sending rate according to perceived congestion levels in the network.

In the absence of ECN (Explicit Congestion Notification) [RF99][HA00], the TCP sender’s primary mechanism to infer packet loss is by observing three duplicate acknowledgements (*dupack*) in the stream of acknowledged packets. As a safeguard for cases where the three-*dupack* scheme does not trigger, TCP employs a retransmission timeout (RTO) mechanism to infer packet loss. Inference of packet loss via RTO results in a more severe performance degradation since the sender cannot send any packets but has to wait for the RTO to expire. In addition, the TCP sender has no choice but to interpret a timeout as an indication of severe network congestion and cuts its sending rate drastically.

There are two complementary approaches to reducing the performance impact of RTO in TCP: (i) modify the TCP flow control scheme so that the *dupack* scheme is triggered more often and there is less reliance on RTO to recover from packet loss (ii) improve the TCP RTO mechanism so that less time is spent waiting for the timer to expire.

This work takes the latter of the two approaches mentioned above. Specifically, it studies the impact of the initial RTO value on the performance of a TCP flow – more details are provided in section 1.2.

In an attempt to make the results as relevant and useful as possible, the author follows Allman and Falk’s recommendations [AF99] regarding approaches to take when studying TCP performance. This includes:

- ✓ Evaluation of the most widely deployed TCP variant - NewReno
- ✓ Consideration of timer granularity used by real TCP implementations

- ✓ **Traffic Models based on the dominant traffic type on the Internet – HTTP**
- ✓ **Experimentation with multiple flows**
- ✓ **Use of performance indicators such as goodput, fairness and loss rates**
- ✓ **Elimination of synchronization effects in the simulation**

For simulation details, please refer to Chapter 7.

1.2 THESIS OBJECTIVES

The objective of this work is to study the impact of the initial RTO value on the performance of TCP flows. The outcome of this work is expected to translate into a recommendation to the IETF Working Group working on standardization of the TCP protocol. Specifically, this thesis accomplishes the following:

- 1. Systematically analyzes the dropping of packets for a single TCP flow. The objective is to identify those packets whose loss would result in a timeout with the initial RTO value.**
- 2. Performs a large-scale study of two-and-a-half hours of Internet Traffic traces from 13 sites. The goal of this exercise is to determine loss rates for those packets that cause timeout with the initial RTO value. Discovering significant loss rates for such packets would provide sufficient motivation for careful selection of the initial RTO value.**
- 3. Experiments with 9 different operating systems to understand the initial RTO value used by TCP implementations on the Internet today.**
- 4. Develops simulation models of HTTP traffic in a RED-based router network.**
- 5. Evaluates the performance impact for a range of initial RTO values using the simulation model developed above. The results of the operating system study are used to determine which initial RTO values to study. The performance impact is assessed for the following cases:**

- ⊗ Low, medium and high loss rates
- ⊗ Different TCP timer granularities
- ⊗ A range of TCP RTT (Round Trip Times)

6. Utilizes a set of defined performance indicators to assess the performance impact.

1.3 THESIS ORGANIZATION

This remainder of this thesis is organized as follows:

- ✓ **Chapter 2** reviews the TCP flow control scheme with specific focus on its RTO mechanism. Related work on the RTO mechanism and initial RTO value is also examined.
- ✓ **Chapter 3** describes systematic experiments with a single TCP flow. The objective of the experiments is to identify specific loss patterns that cause TCP to timeout with the initial RTO value.
- ✓ **Chapter 4** focuses on the Internet Traffic Study. It discusses the topology, dataset and methodology used for the study. The study determines the prevalence of the loss patterns identified in the previous chapter. Chapters 3 and 4 provide the clear motivation for the experiments in Chapters 5, 6 and 7.
- ✓ **Chapter 5** presents results from the experimental study with nine operating systems.
- ✓ **Chapter 6** provides details on the simulation setup including the simulator network topology and traffic models.
- ✓ **Chapter 7** contains results of the simulations. The primary simulations are performed utilizing a long single-run with the same random number seed. We validate this approach by repeating a subset of the simulations using a multiple-repetition completely randomized approach.
- ✓ **Chapter 8** discusses the results of the previous chapters in an attempt to summarize the findings
- ✓ **Chapter 9** draws conclusions, makes specific recommendations related to the initial RTO value and explores possible future study in the area of the TCP RTO mechanism.

Chapter 2 Background and Related Work

This section discusses TCP's flow control mechanisms. It also examines recent work related to TCP's RTO mechanism with special emphasis on research into the initial RTO value utilized by TCP.

2.1 TCP OVERVIEW

TCP (Transmission Control Protocol) is one of the two most commonly utilized Internet transport protocols. It provides flow-controlled, reliable, ordered, end-to-end delivery for application data transfers. The original TCP specification can be found in RFC 793 [Po81]. Eight years later, deployment experience and research feedback resulted in revisions and enhancements to the original TCP specification. These revisions and enhancements were captured in RFC 1122 [Br89]. Together, these two RFCs served as the cornerstone for the definition of TCP.

The TCP specifications were based on some key design principles that have ensured its longevity as the Internet evolved to encompass a range of network topologies and physical media. Its flow control scheme is one example of mechanisms that bear testimony to strength of the original design principles. While there have been various proposals [BA00][KC99][LK00] to improve TCP performance in certain scenarios, majority of the proposals remain true to the original TCP design principles. Some of these corner-stone principles include:

- ❑ Co-operative congestion avoidance
- ❑ Elastic use of available network bandwidth
- ❑ Self-clocking.

The TCP sender marks all outgoing packets with a sequence number so that the receiver can re-create an exact byte-ordered delivery of data to the application layer. The receiver utilizes a cumulative acknowledgement scheme (ACK) [Cla82][All98] to inform the sender of successfully received packets. When the receiver detects "holes" in the stream

of arriving packets from the sender, it generates *ACKs* with the sequence number of the expected packet – this is referred to as a duplicate *acknowledgement* (*dupack*).

2.2 TCP FLOW CONTROL

To prevent “congestion collapse”, the TCP sender flow-controls the amount of traffic it injects into the network. A TCP sender adheres to the flow-control scheme standardized in [AS99] by implementing three key algorithms: slow-start, congestion avoidance and fast recovery.

The sender utilizes a congestion window (*CWND*) to impose a limit on the amount of data that it sends to the receiver in 1 RTT (round-trip time). On startup, TCP starts with a *CWND* equal to 1 segment (recent proposals suggest increasing this to 4Kbytes [AH98][AF98][PN98]). Based on successful *ACKs* from the receiver, the sender effectively doubles its *CWND* with the expiry of every RTT time – this phase of the data transfer is referred to as slow-start (*SS*). The goal of the *SS* phase is for the sender TCP to discover its fair-share network-allowed throughput rate as soon as possible – hence the reason for the exponential growth in sending rate.

On detection of three *dupacks*, packet loss is assumed and the sender halves *CWND*, retransmits the lost packet(s) and enters the congestion-avoidance phase (*CA*). Early versions of TCP (Tahoe) reset *CWND* to 1 on notification of packet loss. Newer versions of TCP (Reno and New Reno) cut *CWND* by half – this is the fast recovery scheme. For purposes of brevity, this work omits finer details of the different versions of TCP but simply notes that discussion on TCP versions Tahoe, Reno, NewReno and SACK can be found in [Hoe95][FF96][AS99][FH99].

The transition from slow-start to congestion avoidance is noted by saving the halved *CWND* value in *ssthresh* variable (slow-start threshold). In the *CA* phase, TCP opens its window linearly so that *CWND* increases by 1 segment size every RTT. The goal of the *CA* phase is to maintain equilibrium for the TCP sender. During equilibrium, on average, a sender should have *CWND* data packets in flight,

where $CWND = BW_{bottleneck} \times RTT$. This is sometimes referred to as the bandwidth-delay product where $BW_{bottleneck}$ is the fair-share bandwidth for this TCP flow at the bottleneck

link and *RTT* is the time between the sending of a data packet by a TCP sender and receipt of an ACK. During the CA phase, if the TCP sender receives three *dupacks* for the same packet, the sender infers packet loss, immediately retransmits the “missing” packet and cuts *CWND* to half. A linear growth of *CWND* then starts again. Figure 1 illustrates the growth and shrinking of *CWND* for a TCP flow with fair-share bandwidth of 400Kbps, 200ms RTT and packets of size 1024 bytes.

The growth and shrinking of *CWND* during the CA phase follows the AIMD (Additive Increase Multiplicative Decrease) scheme. Analysis by Chiu and Jain in [CJ89] suggests that AIMD schemes satisfy the sufficient conditions for convergence to an efficient and fair state regardless of the starting state of the network. The analysis of fairness of AIMD is beyond the scope of this work since it is intrinsic to the implementation of TCP flow control. However, the author notes recent work by Gorinsky and Vin [GV00] who suggest that AIMD fails “to converge to optimal fairness”.

Figure 1 CWND vs Time for a TCP flow

As a backup, in case the three-*dupack* packet loss notification scheme fails, TCP employs a retransmission timeout (RTO). If packet loss does not lead to three-*dupacks* being generated, the sender will be unable to send any more packets as it will have exhausted its allowable *CWND* of packets without receiving *ACKs*. The sender then has to wait for the RTO to expire. On expiration of the RTO, *CWND* is reinitialized to 1 and data transfer resumes with TCP entering the *SS* phase. *CWND* continues to grow exponentially until it reaches the *ssthresh* value at which point *CWND* growth becomes linear. Thus, *ssthresh* is a marker from past-history to tell the TCP sender that it is nearing its equilibrium point and it should become more conservative when injecting packets into the network.

RTOs result in a more severe performance degradation for TCP senders since they stall the self-clocking of new packets and also cause *CWND* to be cut to 1 segment. There has been a lot of recent work into improving TCP's RTO mechanism. This research is examined in the next section.

2.3 TCP'S RETRANSMISSION MECHANISM – RELATED WORK

The area of RTO estimation is an area of TCP that has not received the same level of scrutiny as other TCP flow control mechanisms. Estimating an appropriate value for the RTO is very important. Too small a value may result in needless sender timeout despite the *ACK* being in transit from receiver to sender. Too large an RTO value could result in significantly reduced overall goodput for a flow. Recent results from a particular large-scale Internet Traffic study by Balakrishnan et al [BP98] have shown that approximately 50% of all packet losses require a timeout to recover. Another recent study [LK98] found that over 85% of all timeouts are due to non-trigger of the fast retransmit mechanism. While there are some proposals that seek to reduce TCP's reliance on timeout expiry, these will take time to be discussed, agreed-upon and possibly eventually deployed on a wide-enough scale. In the mean time, there is a clear need for renewed research into the TCP RTO mechanisms.

TCP currently estimates the RTO once per RTT and does not use update its RTO calculation for retransmitted packets – Karn’s algorithm [KP87]. The current scheme for TCP’s RTO estimation was first presented in [Ja88] and is captured in [PA00]:

$$srtt_{new} = (1 - \alpha)srtt_{old} + \alpha \times rtt_{current}$$

$$mdev_{new} = (1 - \beta)mdev_{old} + \beta \times (rtt_{current} - srtt_{new})$$

$$RTO = srtt_{new} + 4mdev_{new}$$

The scheme combines a smoothed weighted average estimate of the RTT ($srtt$) and a mean deviation of the RTT ($mdev$) to obtain the timeout estimate (RTO). RTT samples are obtained by TCP sender implementations by correlating transmitted packets with returned acknowledgements. Karn’s algorithm ensures RTT sanity by requiring the sender to collect RTT samples only for original packets and not for retransmitted packets.

In [AP99], Allman and Paxson undertake a detailed investigation into varied parameter values for α and β . They use a passive analysis method to study Internet traffic from 1995. They conclude that the currently used values of $\alpha=0.125$ and $\beta=0.25$ are reasonably good and the accuracy of the RTO mechanism is “virtually unaffected by ... the settings of the parameters in the exponentially weighted moving average (EWMA) estimators commonly used”. The authors also conclude that the minimum RTO value used plays an important role in the performance of the flow while the timer granularity has less impact and the RTT sampling rate hardly affects RTO estimation. Their study makes no comment on the role of the initial RTO value selected.

Ludwig and Slower [LS00] through analytical modeling and experimentation with long-lived flows in a controlled network, provide results that cause them to disagree with the conclusion of Paxson and Allman. Specifically, the authors claim that as the RTT sampling rate and sender load increase, the choice of parameters for the EWMA estimators is essential to improved RTO estimation. Further, the authors argue that timer granularity needs to be on the order of common RTT found on the Internet today – this is clearly not the case with the 500ms timer as used in many operating systems today. The

authors propose the Eifel retransmission timer as an improvement over the current TCP RTO mechanism.

More recently, Allman, Griner and Richard [AG00] investigate the performance of the retransmission timer over links with varying propagation delay. The authors perform a simulation study with single long-lived flows and no competing traffic. Their conclusion is that the current RTO mechanism sufficiently tracks the RTT of a flow even in the presence of drastic RTT fluctuations. Further, they show that a fine-grained 1-millisecond RTO timer results in reduced flow throughput when compared against a 500-millisecond timer. The authors do not reproduce the above results with short flows or more heavily loaded network links.

In 1998 simulation-based studies, Aron and Druschel [AD98] showed examples of a TCP flow with an RTT of 60-milliseconds and an estimated RTO value of 1.6 seconds. They conclude that the coarse-grained TCP timers are the key reason for such over-prediction and suggest that for the case where the RTT is significantly less than 500-milliseconds, the RTO be fixed to the minimum value of 2 ticks (500 milliseconds to 1 second). In addition they suggest that fine-grained timers (1 millisecond) be used to estimate the RTT while the coarse-grained timers be used to schedule the RTO.

2.4 INITIAL RTO VALUE – EXAMINATION OF RELATED WORK

RFC 1122 [Br89] states that the initial RTO value for TCP should be set to 3 seconds. However, the author of this work has not been able to find any references that investigate or justify selection of that particular value. Further, as subsequent paragraphs will indicate, most TCP implementations did not adhere to this specification for the initial RTO value.

Selection of an appropriate initial RTO value is important. Too large an initial RTO will be problematic for flows with small RTT as it may result in performance degradation in the case of early packet drop. Too small an initial RTO will be problematic for flows with large RTT as it may cause needless retransmissions due to bad timeouts. RFC 2525 [PA99] documents examples where low initial RTO values caused needless performance degradation via early retransmissions.

In Comer and Lin's [CL94] 1994 study of 5 operating systems, none of the surveyed TCP implementations conformed to the 3-second initial RTO. The initial RTO values ranged from 200-milliseconds to 1.5 seconds.

In a subsequent study published in 1997, Dawson et al [DJ97] actively probed the behaviour of 6 commercial TCP implementations and found 5 of the implementations had an initial RTO of 1 second while one of the implementations had an initial RTO of 300-milliseconds.

Paxson undertook a passive analysis study [Pa97b] of 20,000 Internet traces. The traces include data transfers from 8 different TCP implementations including some operating systems previously reported on as part of the studies in [CL94] and [DJ97]. His study confirms that Solaris 2.3 and 2.4 maintain an initial RTO value of 300 msec and suggests that this low RTO value coupled with difficulties in adapting the RTO value to changing RTTs, results in "broken retransmission behaviour". He further finds that the Linux version 2.x TCP implementation uses higher-resolution RTT measurements coupled with more frequent RTT sampling than once per RTT.

Jens Vockler performed detailed tests [JV98] to determine the initial RTO of Solaris 2.5.1 and 2.6 in December of 1998. He experiments with various user settable parameters for a machine running Solaris to understand the relationship between the parameters and the initial timeout value that is seen. The parameter *tcp_rexmit_interval_initial* is settable but does not translate directly into the initial RTO value. His findings are that the initial RTO for Solaris was a multiplicative factor of four times the above parameter plus some unknown constant. He does not state the parameter's default value.

Chapter 3 When is Initial RTO Important?

In studying the effect of initial RTO values on TCP performance, there are two key questions to answer:

1. How often do flows timeout with the initial RTO value
2. When flows experience timeout with initial RTO value, what performance penalty is incurred?

In this chapter and Chapter 4, we attempt to address the first question above. In this chapter we identify specific loss patterns that cause timeout with initial RTO value. In the next chapter, we perform a large-scale study of Internet traffic to determine how often these loss patterns occur. The second question is answered via simulation in subsequent chapters.

3.1 EXPERIMENTAL APPROACH

In this chapter, we perform packet by packet analysis to understand loss situations for which the TCP RTO timer expires with its value set to the initial RTO. Such analysis can be performed theoretically based on the TCP standards. However, we perform the analysis based on experiments using a particular TCP implementation. This is because previous research [Pa97b][DJ97] revealed interesting departures from the expected implementation of the TCP timeout mechanism.

3.2 EXPERIMENTAL METHODOLOGY

The setup and network topology for the experiments is depicted in Figure 2. The key equipment for the experiment consisted of a web browser running on a Linux (kernel version 2.2.12-20) platform, a router on the Nortel Intranet that supported packet filtering and a VxWorks-based (version SENS 5.4) PC with an HTTP web server.

The TCP transfer of the HTTP data is not constrained by Nagle's algorithm or any other such special case mechanisms developed for TCP. Nagle's algorithm is intended to address applications such as Telnet that generate a lot of small packets on the network. Nagle's algorithm essentially says that a TCP connection can only have one outstanding small segment that has not been acknowledged. Since the HTTP server writes its entire 10Kbyte HTML file to the socket at the same time, the TCP stack will not generate any small packets and is not subject to Nagle's algorithm.

Figure 2 Network Topology – Engineered Packet Loss for Linux TCP

In the experiment, the client H retrieves a 10Kbyte file from the HTTP Web Server. The experiments are performed in an Intranet where total transmission delay between client and server is on the order of milliseconds. In the case of a normal transfer with no packet drops, the packet exchange (as illustrated in Figure 2) resembles the following:

1. Client sends a SYN packet to server
2. Server responds with SYNACK packet
3. Client ACKs the SYNACK packet
4. Client sends HTTP "GET" command to server
5. Server ACKs the "GET"
6. Server responds to "GET" with "OK"

7. Client ACKs the “OK”
8. Data transfer phase starts with server sending first data packet
9. After data transfer phase is over, the FIN handshake closes the connection

The 10Kbyte data transfer was repeated numerous times with a particular packet from the transfer being dropped by the router. Specific packet drops were engineered by adding filters to router R based on the source or destination IP address and packet number in the flow. Thus, in the first iteration, a filter was set on the router to drop the SYN packet for that particular 10Kbyte HTML file retrieval. In the second iteration, the filter was changed so the router dropped the second packet – the SYNACK. In the third iteration, the third packet (the HTTP GET) was dropped and so on and so forth. For each iteration, the total transfer time of the 10Kbyte HTML file was recorded and a full packet trace obtained using a Network General Hardware sniffer. Via careful observations, it was possible to determine which packet losses cause timeout with initial RTO value.

3.3 OBSERVATIONS

The key observations from the experimental study can be summarized as follows:

1. Dropping of the SYN Packet causes timeout with initial RTO value – 3 seconds for the Linux client
2. Dropping of the SYNACK Packet causes a timeout with the initial RTO value – 6 seconds for the VxWorks server. The SYN was also retransmitted after an RTO of 3 seconds.
3. Dropping of the GET packet causes a timeout with initial RTO value – 3 seconds.
4. Dropping of a single packet just before or during the FIN handshake caused timeout with value based on the RTO estimation algorithm.

3.4 ANALYSIS

From the previous section, we note that dropping the SYN, SYNACK or the GET packet results in a timeout with initial RTO value. While dropping of the first two packets must necessitate a timeout with initial RTO, we were unsure why the GET timeout relied on the initial RTO. The simple explanation is that the Linux client despite obtaining its first RTT sample on receipt of the SYNACK, does not use this sample to update the smoothed RTT, the smoothed RTT variance or the RTO value. In reality, the first RTT sample that affects the RTO value does not occur until server ACKs the GET packet – this was verified by examining the Linux TCP source code.

The result of losing any of the three packets is a timeout with a value of initial RTO. This causes a 1 second data transfer to be completed in around 4 seconds. In the case of the SYNACK, the total data transfer time is almost 6.7 seconds. This clearly underscores the importance of the initial RTO value. The next chapter investigates the likelihood of losing the SYN and SYNACK packets.

Chapter 4 Internet Traffic Study

This chapter outlines and analyzes a series of studies on Internet traffic traces. The study has two key objectives:

- ❑ To determine per-flow statistics such as packet, byte count and loss distribution on a per-flow basis
- ❑ To determine SYN and SYNACK loss rates for each of the 13 sites.

Figure 3 VBNS Network Topology

While such study could never be considered a complete representation of loss rate in the Internet, it provides a reasonably good indication of the loss rates for SYN[ACK] packets as prevalent in a number of Internet sites. Significant-enough loss rates for these packets would strengthen the need to investigate the parameter to be used as the initial

4.2 DATA SET

A total of 71 individual traces were utilized in the study. Efforts were made to diversify the data set by using:

- a) traces with a range of durations
- b) traces collected on different dates and times
- c) traces collected from major backbone connections in different regions of the US (*west, middle, east*).

Table 1 Description of 71 Data Sets

Dump Sites	# Traces	Trace Duration (s)	Trace Length (N packets)
MaeWest-AIX [NASA-Ames]	5	$10 \leq T \leq 186$	$167568 \leq N \leq 1564405$
Argone Nat Lab (ANL)	6	$103 \leq T \leq 181$	$24637 \leq N \leq 107417$
APAN-(APN) STARTAP	6	$170 \leq T \leq 191$	$95601 \leq N \leq 386898$
Florida U GigaPop (FLA)	5	$160 \leq T \leq 177$	$132914 \leq N \leq 258688$
Front Range Gigapop (FRG)	2	$89 \leq T \leq 184$	$375236 \leq N \leq 539934$
Michigan (MRT) Universities	6	$183 \leq T \leq 193$	$630449 \leq N \leq 1784119$
NCAR – Boulder Colorado	6	$35 \leq T \leq 101$	$25553 \leq N \leq 306946$
North Carolina U Gigapop (NCL)	6	$53 \leq T \leq 169$	$65114 \leq N \leq 339699$
Old Dominion University-ODU	6	$122 \leq T \leq 178$	$120886 \leq N \leq 313395$
Ohio State University-OSU	5	$97 \leq T \leq 189$	$86162 \leq N \leq 889180$
SanDiego State California U-SDC	6	$172 \leq T \leq 192$	$290846 \leq N \leq 767300$
Tel Aviv University- TAU	6	$85 \leq T \leq 177$	$105580 \leq N \leq 554169$
Texas Universities- TXS	6	$68 \leq T \leq 81$	$70061 \leq N \leq 164870$

Table 1 summarizes the 71 individual traces representing traffic from 13 different sites across the US. The geographic location of the 13 sites can be seen in Figure 4. In total, the traces represented 23 million packets, 1.8 million flows and 9600 seconds worth of data. The total number of packets in each trace ranges from tens of thousands to millions while the trace duration ranges between 9 seconds and 190 seconds.

To extract the relevant information from the packet traces, a modified version of the PERL script from [NLAa] was used. A C-based program was then developed to perform the flow management and analysis.

4.3 ANALYSIS METHODOLOGY

There are three possible approaches to loss rate estimation using passive analysis techniques:

- ⊗ Install probes at either the receiver or sender
- ⊗ Install probes at both the receiver and the sender
- ⊗ Install probes at the intermediate routers

The second method listed above is the most desirable method as it would allow for more accurate observation of TCP behaviour. The VBNS traces are obtained using method three listed above. Unfortunately, the trace data does not explicitly capture loss events. Such information can typically only be generated by the routers themselves. Instead, our analysis has to rely on observing the per-packet transmission behaviour to estimate loss. Specifically, the analysis techniques have to rely on observing retransmissions as an approximation to the loss rate. This approach is based on a number of key notions:

1. Recognition that a flow's packets may be reordered on the Internet before arriving at the observing trace node. In addition, some packets of the flow may take an entirely different path to the destination, never passing through the trace node – this would leave holes in the TCP sequence number trail making it difficult to accurately identify retransmissions. To determine how often such scenarios occurred we compared the total bytes observed for a flow against the difference

between the initial and last sequence numbers seen for that flow. We observed that fewer than 5% of the flows had a discrepancy in numbers and in most cases the discrepancy was less than 10% of the total byte count for the flow. Further, the discrepancy affected mostly long-lived flows and was prevalent among flows with more than 2 or 3 Meg of data transfer. As such, it appeared that the scheme of tracking retransmissions would work sufficiently well to approximate loss rate.

2. While identifying retransmissions is useful, it does not sufficiently reflect the number of packet losses. For example: a) broken TCP implementations may retransmit many packets in response to a single packet drop [Pa97b][DJ97] b) certain scenarios may result in valid go-back-N retransmissions. To more accurately reflect packet loss, the metric of “retransmission burst” was utilized. Thus, for the case where N back-to-back packets for a flow are identified as retransmissions, this will only constitute a single retransmission burst. We note encouragingly of a recent study that shows that 62% of web servers use TCP NewReno [PF00]. With New Reno, TCP retransmits multiple lost packets one at a time with RTT gap between retransmissions. This means that the “retransmission burst” metric can suitably serve as an approximation of packet loss. This analysis method can under-estimate the loss rate but not to a large degree because multiple consecutive packet loss for a flow is less common than single packet loss.
3. We do not attempt to track loss of pure ACKs (receiver ACKs with no data) due to difficulties explained later in this section.
4. Detecting retransmitted SYNs and SYNACKs is easy enough since they are the first packets of a flow. However, this information needs to be sanitized to distinguish retransmits due to loss and retransmits due to lack of an end-host to respond to the SYN. The methodology assumes that the probability of routers dropping the same SYN twice is very low. Thus, the corollary assumption that any flow retransmitting more than 2 SYN packets must be faced with a situation where the IP address for the receiver is faulty or the receiver is unable to respond

to the SYN. Such occurrences are not counted as SYN retransmits and are not included in the SYN loss rate statistics.

5. The analysis is based on a one-pass attempt through the trace files and no attempt is made to correlate traces between different sites or even correlate bi-directional flows. This cannot be done because VBNS has altered the trace data to conceal the IP addresses of the traffic traversing the network. As a result of the above methodology, the approach under-estimates the number of retransmissions. Retransmissions that the methodology cannot identify include scenarios such as:

- ☒ If the SYN packet is dropped by a router prior to the link from which the trace was gathered, there is no way to distinguish the retransmitted SYN from the original SYN. The same is true for the SYNACK.
- ☒ If the 2nd and 3rd packets from the sender are both dropped before the trace point
- ☒ If the 3rd sender packet is dropped before the trace point
- ☒ If a subsequent data packet is dropped before the trace point, no more data packets follow and the flow has to rely on a timeout to retransmit the packet.

6. To verify correctness in case of packet re-ordering or alternate path selection, there is need to track flows where the total number of transmitted bytes does not match the difference between the high and low sequence numbers observed (minus sequence numbers consumed by SYNs and FINs). It was verified that majority of these flows did not have any retransmits observed. Such flows with retransmissions did not have the retransmits included as part of the loss rate analysis.

Accurately determining ACK loss is a challenging task that requires correlation of forward and reverse flows in the trace data to even be considered a possibility. Without

this correlation, due to the cumulative ACK scheme, ACK “holes” may be filled by subsequent ACK packets. As such, this study does not attempt to study ACK loss.

Given the above methodology, the author is confident that the analysis performed accurately reflects data packet and SYN, SYNACK packet loss for the trace data studied

4.4 EXPERIMENTAL RESULTS AND ANALYSIS

4.4.1 *Per-Flow Statistics – Observations*

This section presents packet, byte and loss distributions on a per-flow basis. We compare our work against studies from 1998 traffic traces taken from Harvard’s T3 connection to the Internet [Hwa98] and 2 OC-3 connections on the MCI backbone [TM97].

Figure 5 Per-Flow Packet Count Distribution

Figure 5 depicts the per-flow packet count distribution for TCP packets. The results show that majority of the flows (over ninety percent) have fewer than 11 packets. The spike at 25 packets is due to usage of a small set of histogram bins. If the histogram bins had continued to 1000, a heavy-tailed distribution similar to [TM97] and [Hwa98] would be observed. The per-flow byte count distribution shows similar trends as in the [TM97] and [Hwa98] studies. Even though majority of the flows have fewer than 11 packets, there are a few flows with a large number of packets. Detailed examination of

these traces revealed flows with hundreds of packets and millions of bytes. These results are illustrated in Figure 6 – each histogram bin reflects the packet size in bytes divided by 60. e.g. Bin 1 depicts the number of flows with between 0 and 60 bytes, bin 2 depicts the number of flows with between 61 and 120 bytes etc

Figure 6 Per-Flow Byte Count Distribution

Figure 7 depicts the per-flow distribution of retransmission bursts (packet loss). Though not shown in the graph, we note that around 90% of the flows did not experience any packet loss. This high rate of non-lossy flows is somewhat biased by the fact that around 33% of the flows only transmit a single packet – these flows probably had their start or end times corresponding to the ending and starting of the trace respectively. From Figure 7 we observe that a vast majority of the lossy flows experience a single packet drop. The peak at bin 25 reflects those long-lived flows transferring over 1Meg of data. These flows experience a higher number of packet drops as a result of normal TCP congestion avoidance behaviour.

Figure 7 Per-Flow Retransmit-Burst (Loss) Count Distribution

4.4.2 SYN and SYNACK Loss Rates - Observations

In Chapter 3, this work experimentally investigated the impact of SYN and SYNACK loss to the performance of a short HTTP transfer. In this section, we present the results of the Internet loss rate study for SYN and SYNACK packets. The analysis summary for all 67 files is as follows:

- 5% of packets were retransmitted
- The retransmitted burst or loss rate is estimated at 1.3%. The relatively low loss rate compared to previous studies is not unexpected, as the VBNS is a high-performance network. We note that other large-scale Internet studies [Pa97a] have found average loss rates of 5.3%. In addition, on expensive WAN links, packet loss rates of 10-15% are not uncommon.
- 4% of all lost packets were SYN and 2% of all lost packets were SYNACKs
- 16% of flows with loss experienced a SYN or SYNACK packet drop.

The results show that SYN and SYNACK loss is quite significant among packet losses. The high percentage of lossy flows with SYN(ACK) drop suggests *that most flows*

that experience loss are likely to have the first packet dropped. There are a number of possible explanations for such an outcome. One appealing explanation corresponds to the “lockout” phenomenon reported in [QZ99] when short flows startup but find the queue full. Another explanation could be due to the ubiquity of web browsers that start parallel concurrent TCP sessions to retrieve all the objects in a single web page.

Figure 8 SYN and SYNACK Relative Loss Rates

Figure 8 depicts the average relative loss rates of SYN and SYNACK packets for each traffic trace. Relative loss rate is intended to reflect what percentage of the total loss is SYN(ACK) loss. The x-axis corresponds to the 13 sites as listed in Table 1 in descending order. The major findings are:

- ☒ Majority of the traces experienced SYN(ACK) relative loss rates under 10%.
- ☒ A few traces experienced loss rates between 10 and 20%.

Figure 9 shows the 71 traffic traces summary for lossy flows that experience SYN and SYNACK loss. These results are even more staggering than the previous Figure. It confirms the lockout phenomenon - when short-flows startup, they find the queue temporarily full and thus, have the SYN packet dropped.

4.4.3 Broken TCP Implementations

Previous studies [CL94][Pa97b][DJ97] have found various TCP implementations that were inconsistent with the standard IETF specifications. A number of findings of this

study similarly point to the existence of bad TCP implementations. The behaviour of these implementations required a variety of heuristics to be inserted into the loss-rate analysis program that was developed. Without these heuristics, the analysis program would incorrectly report relative SYN loss rates for certain traces.

Figure 9 Lossy Flows with SYN and SYNACK Loss

For example, one particular site (NCL – North Carolina Universities Gigapop) was found to have TCP traces that successfully transferred a small web object with about 8 data packets. On completion of data transfer and receipt of the FIN-ACK, the client retransmitted the SYN packet using the same flow-pair. We rule out the possibility of this being a test TCP stack as the same behaviour was found at this site on 6 different months. In most cases, this TCP stack would not only retransmit the SYN but the entire request-response using the same client port. The same behaviour was seen at the NCL & APN sites. We are unsure of the cause of the SYN re-send and retransmission of all data packets - even those that appear to be ACKed successfully. We understand that TCP implementations sometimes maintain separate queues for SYN and data packets. It is possible that the faulty TCP stack does not properly clear the SYN queue.

Another example of a faulty TCP implementation involved use of the TCP RESET flag. In some traces, the flow sent a packet with the sequence number set to zero and the RESET flag set. The flow then proceeded to go through connection establishment

– sent a SYN, sent an ACK after SYN-ACK receipt, transferred 240 bytes of data and then closed the connection by sending a FINACK followed by an ACK. The TCP standard specification indicates that after a RST, data should not be transmitted on the same flow-pair – the socket should be closed. An example trace exhibiting this behaviour can be found in ANL-950650760 (Feb 15, 2000) – flow 3014.

Other strange behaviour was found for flow 2277 in the above referenced trace. The flow transmitted 4 SYN packets with the same sequence number - 23636045. It then transmitted a fifth SYN packet but changed the sequence number to 23682180. Though the author could not find anything in the TCP standard that prevented this, it is unclear why a TCP implementation would exhibit such behaviour.

Chapter 5 TCP Implementation Study

This chapter presents the results of implementation studies using TCP stacks in nine popular operating systems.

5.1 STUDY OF 9 OPERATING SYSTEMS – WHAT INITIAL RTO IS USED?

For this section, nine different operating systems were experimented with to determine the initial RTO value they used. The objective of the experiment was two-fold. Firstly, to see if the initial RTO values used in TCP stacks today had changed from the values reported in the Related-Work chapter. Secondly, to use these initial RTO values as input to the simulation studies of subsequent sections, thus enhancing the validity and credibility of the simulations.

5.2 METHODOLOGY

Unlike previous studies that used modified active probes or coordinated collection of distributed traces from around the Internet, this study utilizes a different method to estimate the initial RTO. We first identify an FTP server, its gateway router and a group of hosts running the set of operating systems to be studied – these latter hosts are on a separate subnet from the server. At the gateway router for the FTP server, traffic filters are added that cause packets from specific IP addresses to be dropped. The filters match the IP addresses for the hosts with operating systems that we wish to test. An FTP session is then attempted from each of the hosts to the FTP server. The result of the filter match in the gateway router is to drop the SYN packet for the host causing it to be retransmitted again after the initial timeout. The measurements are repeated 10 times to gain confidence in the results. Transmitted packets on the wire are “sniffed” using a Network General Hardware Sniffer (an equivalent tool such as TCPDUMP could also have been utilized). The initial RTO value is recorded along with the exponentially backed off RTO values.

5.3 EXPERIMENTAL RESULTS – INITIAL RTO VALUE

Table 2 presents the average initial RTO value measured for the nine operating systems. In addition, the 95% confidence interval is also presented. The confidence interval is obtained using the method outlined by Banks and Carson in [BC84] for small sample sizes: $\bar{X} \pm t_{\alpha/2, N-1} \left(\frac{S}{\sqrt{n}} \right)$ where $\bar{X}$ is the mean of the sample, S^2 is the sample variance, n is the sample size and $t_{\alpha/2}$ is the interval coefficient for the t distribution table ($N - 1$ degrees of freedom). The interval coefficient value for the 95% confidence interval is 2.145.

Table 2 Average initial RTO Values

Operating System	Average Initial RTO Value	95% Confidence Interval
Win98	2.97s	± 0.02064
WinNT 4.0 – Serv Pack 3	2.98s	± 0.028933
Linux 2.2.12-20	3.0s	± 0.002896
Sun Solaris (SunOS 5.7)	3.37s	± 0.012807
SunOS 4.1.4 1	5.7s	± 0.122458
Irix 6.2	5.82s	± 0.154216
VxWorks 5.4 – SENS	5.86s	± 0.119714
NetBSD 1.0A	5.81s	± 0.072211
HP-UX B.10.20	4.22s	± 0.85118

5.4 OBSERVATIONS AND ANALYSIS

Key observations from Table 2 are:

- Three operating systems - Win98, WinNT and Linux - have initial RTO values that are virtually identical to what is suggested as a “SHOULD” in the TCP standards – the confidence intervals for these three operating systems suggest adherence to the TCP standard.

- ❑ Four of the operating systems - **SunOS 4.1.4, IRIX 6.2, VxWorks 5.4** and **NetBSD** - all have initial RTO values that are almost *twice* the recommended value in the standards.
- ❑ The **HP-UX** initial RTO value fluctuated greatly through the various tests as evidenced by the relatively large confidence interval.
- ❑ The **Solaris** initial RTO was found on average to be around 3.3 seconds. Previous work [CL94] found this value to be around 300 milliseconds.

Since the previous studies on initial RTO value, there have clearly been TCP implementation changes to this value. However, there still remain a number of implementations with initial RTO values that are clearly too high and will result in performance degradation of short flows because of unnecessary idle time when early packets are dropped. In subsequent sections, via simulation and live network experimentation, we try to understand the impact of choosing such initial RTO values.

5.5 EXPONENTIAL BACKOFF RESULTS

In addition to obtaining the empirical value of the initial RTO used in the operating systems, we also recorded the exponential back-off values used by the systems when the sender incurs multiple sequential timeouts for the same packet. This information has no bearing on the work of this thesis and is only captured here to reflect the disparity between the TCP standard recommendation and the actual implementations realized in popular operating systems.

The average exponentially backed off RTO values are illustrated in Figure 10. The key observations can be summarized as follows:

- ❑ Three operating systems - **Win98, WinNT** and **Linux** - have exponentially backed off RTO values that are virtually identical to what is suggested as a **SHOULD** in the TCP standards – they start at three seconds and double with each successive backoff.
- ❑ Three operating systems (**NetBSD, SunOS 4.1.4** and **SunOS 5.7[Solaris]**) exhibit RTO values that double with each backoff – despite the fact that the initial RTO value is not in keeping with the standard suggestion.

- Three operating systems (HP-UX, VxWorks and Irix) appear to use an alternative backoff multiplier other than 2. Some of the alternative multipliers appear to be quite large

Figure 10 Exponentially Backed Off RTO Values for 9 Operating Systems

Chapter 6 Simulation Setup

In this chapter, we utilize a simulation study to investigate the effect of the initial RTO value on the performance of TCP flows. The studies focus on short TCP New Reno flows in a RED-based router network. Recent studies [CM98] and [MC00] have shown that 83%-95% of all IP traffic is TCP and 65%-70% of the TCP packets are HTTP. Thus, a minimum of 50% of network traffic consists of short flows. Further, [CM98] reports a study on packet count per flow. The results reveal a distribution that is heavy-tailed with an average under 20 packets. Quite clearly, despite the growth in recent UDP multimedia applications and TCP NAPSTER applications, short TCP flows continue to dominate the Internet traffic and thus, deserve special attention to improve their performance.

In [PA00], the authors specify that the RTO timer value should reflect the initial RTO value (and exponentially multiplied values in the case of backoff) until the first RTT measurement is made. At that point, the initial RTO value should be discarded and the RTO should be strictly dependent on RTT measurements and the RTO calculation formula specified in [PA00]. It is unclear whether most operating systems today actually follow this recommendation in [PA00] or whether early RTO calculations decay the initial RTO value. In the latter case, the choice of initial RTO value is even more important as it could have a larger performance impact. In this study, we do not pursue the latter possibility – we have been unable to find any studies that indicate how the popular operating systems actually utilize the initial RTO value. Instead, we perform the study using the method where the initial RTO value is only good until the first RTT measurement. We assume that most operating systems will move to this method in the future.

6.1 NETWORK TOPOLOGY

The *ns-2* simulator [NS] is used for the simulation study. The key benefit of utilizing this simulator is that *ns-2* provides models of TCP, RED-enabled routers that

closely emulate the implementations in deployed systems today. The network topology utilized for the study is depicted in Figure 11. The traffic consists of a mixture of long-lived FTP flows and short-lived HTTP-like flows.

Figure 11 Simulation Network Topology

The topology consists of 10 web servers and 10 web clients. The web clients access the web servers through a network with a bottleneck link of C_2 Mbps and transmission delay of T_2 ms. The leaf routers (L_1 and L_2) emulate routers inside an Intranet or MAN and thus connect to the web servers over links with 10Mbps bandwidth and smaller transmission delays. The routers bounding the bottleneck links (R_0 , R_1 , R_2) use RED for buffer management while the leaf routers and other nodes use Drop-Tail buffer management. The bottleneck link is shared by both web traffic as well as long-lived FTP traffic. We ensure that there is no packet loss on the access links between the web servers and the leaf routers or between the leaf routers and the bottleneck link routers.

Relevant simulation parameters can be found in Table 3. The parameters were used for all experiments unless explicitly stated otherwise.

Table 3 Simulation Parameters

Parameter	Value
TCP Sender	New Reno
TCP Receiver	Delayed ACK
RED Q Parameters (minth, maxth, maxp, wq)	40pkts, 80pkts, 0.1, 0.002
DropTail Q Sizes (Bottleneck, Access Links)	100 pkts, 200 pkts
Packet Size	1000bytes
TCP Timer Granularity	0.5s
TCP Receiver Max Window	500 pkts
# FTP Flows	Depends on testcase
# HTTP Flows	See next section

6.2 TRAFFIC MODEL – BACKGROUND FTP FLOWS

To create congestion for both the HTTP GET requests and the HTTP data, bi-directional background traffic is also generated in the form of infinitely long-lived FTP flows. The HTTP data shares the same congested link (R_0 - R_2) as FTP data flowing from FS_2 to FC_1 – the FTP client. The HTTP requests share the same congested link (R_1 - R_2) as FTP data from FS_1 to FC_1 . Depending on the particular testcase, the number of background FTP flows is varied. The start times of the FTP flows are uniformly distributed between 0 and 1 second.

6.3 TRAFFIC MODEL – HTTP FLOWS

The web traffic model in the ns-2 simulator is described in some detail in [FG99]. This same model is used for the first experiment described in the next chapter. In this section, we review the salient features of this model.

The current implementation in the simulator uses the single direction TCP model – in this model, the three-way opening TCP handshake is omitted and data can only be transferred uni-directionally with only ACKs returnable on the reverse direction. This is slightly different from the real-world where the client would send a SYN pkt, the server would respond with a SYN-ACK and then the client would respond with an ACK followed by a GET message. The server would then ACK the GET, respond with an OK and then commence data transfer – all on the same flow-pair. To more closely emulate the real-world, the ns-2 implementation utilizes two separate flow-pairs. A single flow-pair is used to send a GET from client to server for which it receives an ACK. As soon as the ACK arrives at the client, the server begins the data transfer.

The ns-2 HTTP model allows flexibility as far as the following elements:

- ✓ Number of Sessions
- ✓ Number of Pages per Session
- ✓ Page Size – allows configuration of number of objects per page. Can be a constant or a function of a probability distribution function
- ✓ Inter-Page Time – the time between the user clicking to retrieve new pages. Various probability distribution functions can be used.
- ✓ Object Size – the number of packets in a particular object. Various probability distribution functions can be used.
- ✓ Inter-Object Time – the time between the server sending all the objects in a page. This reflects stack and socket delay time. Again various probability distribution functions can be used.
- ✓ Page Location – the pages accessed by a single session are uniformly distributed amongst all the defined servers in a server pool. For the purposes of this study, there's only a single server pool.
- ✓ Object Location – all objects for a page are accessed from the same server but transmitted on different flows.

The HTTP traffic used for the first experiment consists of 10 sessions each with 400 pages transferred. For each new page, the client randomly selects a server from the pool. Inter-page generation times for a session are exponentially distributed with an average of 3 seconds. Each page consists of a number of objects. All objects in a page are sent from the same server to the client. The number of objects per page is exponentially

distributed with an average of 2. A constant object size of 8 packets is used even though a Pareto distribution is often used to model this parameter. This is done to facilitate easy comparison of HTTP transfer delay in various simulations. If different object sizes are used, it becomes more difficult to calculate performance indicators on a per-object basis. The inter-object generation time has an exponential distribution with an average of 0.01s to model delays at the server. Each object starts a new TCP flow.

Table 4 depicts the simulation parameters used for the web traffic model. The parameters are based on various studies of Internet HTTP traffic. One such study by Bruce Mah [Ma97] revealed a mean object size of 8-10Kb and a mean page size of 2.8 to 3.2 objects. The mean page size is further confirmed by studies in [CW95] where Claffy et al find that 70-80% of the flows have less than 10 packets. Feldman et al [FR98] find that the mean file size is 16 packets with an average size of 612bytes (a total of around 9.6Kbytes). Finally, Abrahamsson in [Ab99] finds an inter-page average of 1 second.

The simulation runs for about 1300 seconds. Approximately 10,000 objects were generated over the duration of the simulation. Sessions are started 50 seconds after the long-lived FTP flows are started. The 10 sessions are started sequentially with inter-session start times uniformly distributed between 0 and 2s. The same random seed was utilized for all the experiments. Utilizing different random seeds would change the traffic model and make comparison on a per-object basis more difficult and less conclusive.

Table 4 HTTP Traffic Model Parameters

Parameter	Value
# Sessions	10
# Pgs/Session	400
InterPage Time	Exponential Distribution Avg = 3s
# Objects Per Page	Exponential Distribution Avg = 2
Inter-Object Time	Exponential Distribution Avg = 100ms
Object Size	8pkts

The same random seed is utilized for the different simulations in order to reproduce the traffic model for all experiments. i.e we use a deterministic approach with

the random number generator. In section 7.6, we repeat some of the experiments using a completely randomized approach with multiple repetitions per experiment. We find the results of the single-long-run deterministic approach to be within the 95% confidence interval of results for the single-factor-completely-randomized approach.

Chapter 7 Simulation Results

This chapter presents results of simulation experiments using short flows with particular focus on the impact of initial RTO value. We define a number of performance indicator metrics, which are utilized to compare the effect of different initial RTO values. The experiments are performed in the context of different factors such as:

- Effect of Loss Rate
- Effect of RTT size
- Effect of TCP Timer Granularity

7.1 PERFORMANCE METRICS

To compare results between various simulations we utilize a set of metrics related to the transfer time for an object. We define Transfer Time T_i for object i to be:

$$T_i = T_{LAST_ACK} - T_{GET}$$

where T_{GET} is the time when the client sends the GET packet to the server and T_{LAST_ACK} is the time when the sender receives the ACK for the last packet of data for object i . It could also have been possible to ignore the time for the GET and its ACK in calculating the transfer time for an object. However, since the delivery of the object data is dependent on arrival of the GET, we include it in the transfer time calculation. Further, since the ns-2 model does not include the full-three way handshake, including the GET causes the transfer time to more closely resemble the transfer time that will be seen in a real network. Calculating the transfer time using the above method requires correlation and identification of the two separate flow-pairs utilized by ns-2 to represent the GET-request flow and data-transfer flow.

We use the transfer times for a simulation to compute three performance indicators that assist in evaluating the effect of various initial RTO values: average transfer delay, standard deviation of the transfer delay, fairness.

Average Transfer Time

The average transfer time T_{avg} for N objects is a useful metric because it will compare two simulations which are identical in their generation of HTTP objects but whose transfer times are affected due to use of different initial RTO values. This metric is computed as follows:

$$T_{avg} = \frac{\sum_{i=1}^N T_i}{N}$$

Standard Deviation of the Transfer Time

The second performance metric utilized is the standard deviation of the transfer times for the objects. This is measured over the entire sample. The standard deviation is useful because to some extent it will indicate the bias of certain initial RTO values against certain flows and not others. While long-lived FTP flows will generally share link bandwidth fairly (if RTT, packet size etc are equal), this is not true for short flows as most of these flows terminate while they are still in the Slow Start phase. Generally, it is expected that transfer times for such flows will show a greater variance or standard deviation than for long-lived flows. However, since we compare standard deviation values between simulation runs where TCP flows have different initial RTO, we are only interested in the relative comparison of standard deviation of transfer times as opposed to the actual values.

$$T_{std} = \sqrt{\frac{\sum_{i=1}^N (T_i - T_{avg})^2}{N}}$$

The author notes that the relative standard deviation could also be used as a variability measure. However, since the comparison is only made for cases where the mean is relatively similar, the standard deviation is taken as a suitable metric. The relative standard deviation could be computed as follows:

$$T_{relative-STD} = \sqrt{\frac{T_{std}}{T_{avg}}}$$

Fairness Index for Transfer Times

The final metric is the fairness index (FI) [Jai91]. The FI is a measure to determine how fair is the treatment given to individual users by the network. We use this metric to represent the variability in transfer times for various objects. The FI is

computed as follows:
$$FI = \frac{\left(\sum_{i=1}^N T_i \right)^2}{N \sum_{i=1}^N T_i^2}$$

The FI lies between 0 and 1. A higher value for FI indicates greater fairness or lesser variability amongst object transfer times. A value of 1 for FI indicates that all objects have the same transfer time

7.2 TRAFFIC MODEL – HTTP FLOWS

The HTTP traffic model utilized for the first experiment was described in the previous chapter. In this section, we characterize the generated traffic to understand how it compares to real-world measurements and the desired model.

Figure 12 Page Size Distribution

The traffic model described in section 6.3 had 10 HTTP sessions consisting of 400 pages each. A total of 10,000 objects were generated with 8 packets per object and 1000 bytes per packet. Figure 12 plots the frequency distribution of page sizes that were generated by the HTTP traffic model used in the experiments. From the Figure we note

that most of the pages have between 1 to 4 objects with an average of 2.45 objects per page. The average for the exponential distribution was set to 2.

Figure 13 plots the measured inter-page generation time for the first simulation experiment. Again the Figure reveals an exponential distribution with an average of 2.94 seconds between the generation of each page – the specified average for the model was 3 seconds.

Figure 13 Inter-page think-time distribution

Figure 14 plots a histogram of the inter-object start times. The distribution follows that of an exponential curve with an average of 10.93 milliseconds – the model average was specified as 10ms.

Figure 14 Inter-object time distribution

7.3 IMPACT OF LOSS RATE

In this section we investigate the effect of the initial RTO value in impacting the transfer delays of web traffic under low, medium and high loss rates. The network setup is described in Chapter 6. The values for network parameters C_1 , C_2 , C_3 , T_1 and T_2 are set to 10Mbps, 10Mbps, 100Mbps, 5ms and 10ms respectively. We select loss rates of 2.2%, 6.4% and 9.5% to reflect low, medium and high respectively. Background FTP flows are utilized to increase bottleneck link congestion to sufficient levels that can realize the desired loss rates – 8, 25 and 40 FTP flows were utilized for the increasing loss rates respectively. We evaluate the performance metrics defined earlier in this paper to analyze the effects of the initial RTO value.

Figure 15 Initial RTO Effect for Different Loss – Average Delay

7.3.1 Experimental Results and Observations

Figure 15, Figure 16 and Figure 17 illustrate the effect of initial RTO values on the average, standard deviation and fairness of transfer delay respectively – for three different loss rates. The initial RTO values considered reflect the values currently used in various TCP stacks deployed on the Internet today – section 5.3.

Figure 16 Initial RTO Effect for Different Loss – Delay Standard Dev

The key observations from the three Figures include:

- ✓ For a loss rate of 2.2%, as the initial RTO increases from 0.5s to 6.5s, the average transfer delay increases from 3.92s to 4.32s (a 10% increase); the standard deviation of the transfer delay increases from 1.45 to 2.68 – (an 84% increase); the fairness index decreases from 0.88 to 0.73 – (a decrease of 20% decrease).
- ✓ For a loss rate of 6.4%, as the initial RTO increases from 0.5s to 6.5s, the average transfer delay increases from 6.63s to 8.43s – (a 27% increase); the standard deviation of transfer delay increases from 4.9 to 8.01 – (a 63% increase); the fairness index decreases from 0.65 to 0.52 – (a 20% decrease).
- ✓ For a loss rate of 9.5%, as the initial RTO increases from 0.5s to 6.5s, the average transfer delay increases from 8.4s to 13.6s – (a 61% increase); the standard deviation

of transfer delay increases from 7.8 to 21 – (a 169% increase); the fairness index decreases from 0.54 to 0.28 – (a 48% increase).

Figure 17 Initial RTO Effect for Different Loss Rates –Transfer Delay Fairness

7.3.2 Analysis

In this section, we assess and analyze the results of the previous section. Firstly, we observe that for low loss rates the average transfer delays don't vary much regardless of the initial RTO value selected. This is because there are probably not very many packets being dropped that cause timeout with initial RTO value. Secondly, we notice that for higher loss rates, the average transfer delay varies to a greater extent as the initial RTO value is increased. In this scenario, we suspect greater dropping of SYN, SYNACK and HTTP GET packets. Finally, we notice that the initial RTO values of 0.5s and 1s provide the lowest average transfer delay, and best fairness index – with an initial RTO of 1s providing better fairness and a lower standard deviation than 0.5s. We are unsure if this deviation is because of simulation anomaly or a reflection of network conditions due to more aggressive TCP sender behaviour.

7.4 TCP TIMER GRANULARITY

In this section we investigate the effect of the initial RTO value on performance for different granularities of the TCP tick timer. A total of 20 background FTP flows are generated. The RTT for the HTTP flows is around 75ms, reflecting common RTT values used on the Internet [FP98]. The loss rate on the bottleneck link is approximately 2.45% with no loss on other links. The network setup is described in Chapter 6. The values for network parameters C_1 , C_2 , C_3 , T_1 and T_2 are set to 10Mbps, 10Mbps, 100Mbps, 5ms and 10ms respectively.

Figure 18 Initial RTO Value Effect for Different Timer Granularities–Avg Delay

7.4.1 TCP Short Flow Traffic Model

The short flow traffic model used in this section and the next, differs slightly from the one used in the previous section. Here, the model of short TCP flows is not exactly HTTP-like but will reflect the performance impact on TCP flows that rarely exit the Slow-Start phase of data transfer. Such a model is applicable not just to HTTP traffic but other short-lived TCP flows such as SMTP or other query-response applications. Essentially, we attempt to mimic the NETPERF tool [NETP]. This models transactional transfers by sending a small request and getting a response from the server before sending the next request. Thus, transactions are executed sequentially with each new transaction

transmitting only after the previous transaction for its session has completed. To reduce synchronization effects, we introduce a small time between the end of one transaction and start of the next – the duration of this time is uniformly distributed between 0 and 1second. The sources and destination nodes remain the same as in the previous experiment.

We mimic 10 HTTP sessions continually retrieving back-to-back transactions or objects with each object utilizing a distinct TCP flow. We fix the object size to 8 packets as in the previous experiment. The packet size is 1000 bytes. A total of 10,100 objects are generated.

7.4.2 Experimental Results and Observations – Timer Granularity

The test is repeated for TCP timer granularities of 50ms, 100ms and 500ms. TCP implementations in most deployed operating systems utilize a 500ms timer. However, a number of implementations such as Linux utilize more fine-grained timers. Note: for test with 500ms, the loss rate was low at 1.74%. Higher loss rate would have exacerbated the results. Also, 25 FTP flows were used for the test with 100ms timer while 20 FTP flows were used for the test with 50ms and 500ms timers.

Figure 19 Initial RTO Value Effect for Different Timer Granularities – Std Dev

Figure 18, Figure 19 and Figure 20 illustrate the effect of initial RTO values on the average, standard deviation and fairness of transfer delay respectively – for three different granularities of the TCP tick timer. The key observations are:

- ✓ For a granularity of 50ms, as the initial RTO increases from 0.5s to 6.5s, the average transfer delay increases from 0.85 to 1.21 (a 39% increase); the standard deviation of the transfer delay increases from 0.54 to 2.11 (a 290% increase); the fairness index decreases from 0.71 to 0.24 (a 66% decrease)
- ✓ For a granularity of 100ms, as the initial RTO increases from 0.5s to 6.5s, the average transfer delay increases from 0.92 to 1.27 (a 38% increase); the standard deviation of the transfer delay increases from 0.55 to 2.33 (a 320% increase); the fairness index of the transfer delay decreases from 0.73 to 0.23 (a 68% decrease)
- ✓ For a granularity of 500ms, as the initial RTO increases from 0.5s to 6.5s, the average transfer delay increases from 0.94 to 1.16 (a 23% increase); the standard deviation of the transfer delay increases from 0.76 to 1.81 (a 138% increase); the fairness index of the transfer delay increases from 0.61 to 0.29 (a 52% decrease)

Figure 20 Initial RTO Value Effect for Different Timer Granularities – Fairness

7.4.3 Analysis – Timer Granularity

In this section, we assess and analyze the results of the previous section. We observe that for all three timer granularities, initial RTO values of 0.5s and 1s result in the smallest average and standard deviation of the transfer delay. Further, the initial values of 0.5s and 1s result in the best result for the fairness index. There is also a clear improvement in performance (for all three metrics) as the initial RTO is decreased from 6.5s to 0.5s. The few anomalous results are attributed to simulation error.

7.5 IMPACT OF RTT SIZE

In this section, we investigate the initial RTO value's effect on short flow transfer delay – for a range of RTTs. It is difficult to generalize mean or median RTTs for the Internet since this is dependent on topology and transmission media capabilities. As such, we restrict our studies to RTT values that are reportedly common in the Internet, and for borderline large RTT values such as:

☒ RTTs for satellite links

☒ RTT for slow modem links with large buffers on the network side.

7.5.1 Borderline Large RTT experiment

In this section, we focus on experiments using some of the larger RTT values found on the Internet. We use 500ms to reflect a typical satellite RTT. We use 1s, 3s and 5s as RTT values on modem dialup links during busy time (we have verified this on a 56Kbps modem). The traffic model is the same one used for the test with TCP timer granularity. The number of background flows for this test is 40. We initiate a set of tests using the 0.5s TCP timer granularity to reflect what is used in most TCP implementations today. The network setup is described in Chapter 6. The values for network parameters C_1 , C_2 , C_3 , and T_2 are set to 10Mbps, 10Mbps, 100Mbps, and 20ms respectively. The value for T_1 is varied as required to reflect the desired RTT for the TCP session.

Table 5 Avg Transfer Delay for Different RTTs (s)

Initial RTO Value(s)	RTT=0.25s	RTT=0.55s	RTT=1s	RTT=3s	RTT=5s
0.5	1.89	3.04	6.41	18.06	27.3
1	1.89	3.06	6.58	18.69	29.7
1.5	1.93	3.08	5.58	18.43	30.47
2	1.96	3.09	5.64	18.69	29.6
2.5	1.98	3.09	5.63	18.99	30.11
3	2.03	3.17	5.7	18.99	30.4
3.5	2.08	3.24	5.72	16.12	30.68
4	2.1	3.23	5.73	16.24	30.66
4.5	2.08	3.25	5.73	16.09	30.86
5	2.27	3.36	5.94	16.16	30.84
5.5	2.24	3.35	5.92	16.28	26.03
6	2.24	3.41	6	16.36	26.4
6.5	2.27	3.44	5.98	16.3	26.1

7.5.2 Large RTT – Observations

Table 5, Table 6 and Table 7 capture the average, standard deviation and fairness index for the transfer delays of the short TCP flows in the experiments. From the tables, the following key observations can be made:

- ⊗ There is at most a 10% difference in average transfer delays between any two initial RTO values for the large RTT values of 1s, 3s and 5s. However, for the RTT of 250ms this difference approaches 30%. This indicates that for flows with large RTT and small initial RTO, there is at most a 10% penalty in performance. In addition, as the RTT reduces to more common values (250ms), there is a clear performance penalty in selecting a larger initial RTO value.

- ⊗ The standard deviation for this set of tests reflects the previously observed trend of increasing as the initial RTO value is increased. However, for this particular testcase we suggest that it is not that useful a metric as it does not reflect the key indicators that we wish to observe.
- ⊗ The fairness index also shows decreased values as the initial RTO value is increased. Further, we observe that for RTT values of 1, 3 and 5s, the maximum difference in fairness is approximately 10%. However, when the RTT approaches 250ms, the fairness index varies by 40% between its minimum at initial RTO of 0.5s and its maximum when the initial RTO is 6.5s. This is good because it indicates that extra traffic due to small initial RTO does not result in significant performance penalty.

Table 6 Standard Deviation for Different RTTs

Initial RTO Value(s)	RTT=0.25s	RTT=0.55s	RTT=1s	RTT=3s	RTT=5s
0.5	1.16	1.51	2.09	4.78	7.97
1	1.28	1.65	2.32	6.29	8.67
1.5	1.28	1.57	2.63	6.44	10.1
2	1.34	1.7	2.71	5.74	8.04
2.5	1.49	1.69	2.69	6.38	9.24
3	1.83	2	3.07	6.33	9.75
3.5	1.74	2.28	3.04	7.14	10.1
4	2.1	2.04	2.99	7.46	9.68
4.5	1.88	2.15	2.86	6.91	9.88
5	3.04	2.61	3.56	7.14	9.65
5.5	3.09	2.62	3.44	7.22	10.98
6	2.84	3.38	3.73	7.84	11.55
6.5	2.56	2.99	3.88	7.4	11.3

Table 7 Fairness Index for Different RTTs

Initial RTO Value(s)	RTT=0.25s	RTT=0.55s	RTT=1s	RTT=3s	RTT=5s
0.5	0.72	0.8	0.9	0.93	0.92
1	0.69	0.77	0.89	0.9	0.92
1.5	0.69	0.79	0.82	0.89	0.9
2	0.68	0.77	0.81	0.91	0.93
2.5	0.64	0.77	0.81	0.9	0.91
3	0.55	0.72	0.78	0.9	0.91
3.5	0.59	0.69	0.78	0.84	0.9
4	0.5	0.71	0.79	0.83	0.91
4.5	0.55	0.69	0.8	0.84	0.91
5	0.36	0.62	0.74	0.84	0.91
5.5	0.34	0.62	0.75	0.84	0.85
6	0.38	0.5	0.72	0.81	0.84
6.5	0.44	0.57	0.7	0.83	0.84

During this experiment, we observed the number of timeouts for the transferred objects. As expected, when the initial RTO value is smaller than the RTT, there will be many retransmissions because the sender retransmits too early. This can be seen from Table 8. Interestingly enough, despite the large increase in timeouts for small initial RTO and RTTs of 3s and 5s, this is not reflected in terms of a significant increase in average delay – as mentioned previously, there is a difference of at most 10%.

7.5.3 Large RTT – Analysis

Based on the results, we conclude that there is only a small performance penalty due to unnecessary retransmissions when the RTT is larger than the initial RTO. The set of experiments resulted in a maximum penalty in average delay of 10%. Since such borderline large RTT values are not very common on the Internet, it implies that the initial RTO value can probably be set at some value smaller than the RTTs experimented with.

Table 8 Number of Timeouts – Varied RTT

Initial RTO Value(s)	RTT=0.25s	RTT=0.55s	RTT=1s	RTT=3s	RTT=5s
0.5	3362	3371	23026	32822	42494
1	3298	3318	13617	23486	32289
1.5	3312	3362	3403	22717	22613
2	3294	3254	3473	13383	22375
2.5	3177	3064	3404	13510	22066
3	3177	3228	3433	13383	12836
3.5	3316	3299	3402	3369	12805
4	3170	3233	3357	3421	12780
4.5	3116	3182	3322	3305	12728
5	3337	3329	3640	3305	12637
5.5	3217	3230	3495	3406	3287
6	3143	3225	3594	3388	3524
6.5	3208	3245	3464	3322	3312

7.5.4 Common Internet RTT and Small TCP Tick Timer - Experiment

In this section, we present results from a set of experiments with smaller RTT and more granular TCP timer that can accurately reflect the correct RTT and RTO. The experiment is performed for short TCP flows with an RTT of 80ms and a TCP tick timer granularity of 50ms. The experimental setup is the same as in section 7.5.1 except for the T_2 parameter, which is set as required to obtain the 80ms RTT.

Figure 21 Average Delay – Common RTT; Small Timer

7.5.5 Common RTT – Observations

The results of the experiment with the common RTT can be observed in Figure 21, Figure 22 and Figure 23. From the Figures, we observe the following:

- ☒ There is a marked increase in average and standard deviation of the transfer delay as the initial RTO value is increased from 0.5 to 6.5. The average delay increases by 71% while the standard deviation increases by 460%.
- ☒ There is a marked decrease for the fairness index as the initial RTO increases – the FI decreases from 0.63 to 0.14 – a decrease of 77%.

Figure 22 Standard Deviation – Common RTT; Small Timer

7.5.6 Common RTT - Analysis

Based on the results in the previous section, we observe a performance gain of 70% for average delay or the fairness index when an initial RTO value of 0.5s or 1s is used instead of 6.5s for the initial RTO value. Further, the standard deviation of the transfer delay is lowest for initial RTO value of 0.5s and 1s. The performance gain is due to the TCP flow not waiting unnecessarily for a large initial timeout period.

Figure 23 Fairness Index – Common RTT; Small Timer

7.6 VALIDATION OF SIMULATION METHODOLOGY

The previous sections of this chapter presented simulation results based on a single simulation run with a large number of sampled data transfer times. The same random seed was utilized for all simulations in order to facilitate comparison. In this section, we validate the approach by comparing its results against another simulation approach. In this new approach, we repeat one of the experiments using different random seeds, a shorter number of sampled data objects and a larger number of simulation runs. We use the ANOVA (Analysis of Variance) technique to determine effect of the factor and to develop confidence in the results.

7.6.1 Single-Factor Completely Randomized Experimental Design

In this section, we consider a single-factor completely randomized experimental design. We utilize the scheme as described in [BC84]. A single-factor experiment is one where there is only one factor having different k different levels. In our case, the factor is the initial RTO value and it has 13 different levels. In the experiments to be described, the replications of the model at each level and for different levels of the factor all utilize independent streams of random numbers – this is referred to as *completely randomized design*. The effect of level j of the factor is called τ_j .

The statistical model of the analysis of single-factor completely randomized experimental design with k treatment levels is:

$$Y_{rj} = \mu + \tau_j + \varepsilon_{rj}$$

where $(r = 1, 2, \dots, R_j; j = 1, 2, \dots, k)$ and Y_{rj} is the observation r of the response variable for level j of the factor; μ is the overall mean effect, and ε_{rj} is a random error in observation r at level j ; R_j is the number of observations (replications) made at level j . The model of the above equation assumes that the responses (Y_{rj}) at level j vary about the mean $(\mu + \tau_j)$ for that level by some random error (ε_{rj}).

To test that the levels of a factor have an effect on the response, the initial analysis performs a statistical test of a particular hypothesis. The tested hypothesis is that

the levels of the factor have no effect on the response. If the hypothesis is not rejected, one can safely conclude that the factor has no significant effect on the response variable. If the hypothesis is rejected, the analyst may conclude that the levels of the factor do have some effect upon the mean response. The hypothesis tested is:

$$H_0 : \tau_j = 0 \text{ where } (j=1,2,\dots,k)$$

To test the above hypothesis H_0 , we perform the one-way ANOVA (Analysis of Variance) test using an F statistic of the hypothesis in the above equation. Using samples for all levels and replications, we first compute $\bar{Y}_{..}$ (the overall average of all samples) and $\bar{Y}_{.j}$ (the average of all samples for level j of the factor). We then compute:

$$SS_{TOTAL} = SS_{TREAT} + SS_E$$

where

$$SS_{TOTAL} = \sum_{j=1}^k R_j \left(\bar{Y}_{.j} - \bar{Y}_{..} \right)^2 + \sum_{j=1}^k \sum_{r=1}^{R_j} \left(Y_{rj} - \bar{Y}_{.j} \right)^2$$

Finally, we compute:

$$F = \frac{MS_{TREAT}}{MS_E} = \frac{SS_{TREAT} / (k-1)}{SS_E / (R-k)}$$

where R is the total number of samples and k is the number of levels for the factor. Thus, when hypothesis H_0 is true, the above test statistic has an F distribution with $(k-1)$ and $(R-k)$ degrees of freedom. The ANOVA test of the hypothesis H_0 is to reject H_0 if $F > F_{1-\alpha}$ where $(1-\alpha)$ is the probability of an F statistic with $(k-1)$ and $(R-k)$ degrees of freedom being below the critical value $F_{1-\alpha}$.

7.6.2 ANOVA – Approach and Experimental Setup

In this section we present the results of the single factor completely randomized test and the resultant ANOVA analysis. We do not repeat this test for all the experiments in earlier sections of this work. Rather, we repeat solely for a single experiment – the effect of initial RTO on performance for a TCP timer granularity of 50ms. Specifically,

we repeat the experiment of section 7.4 to ascertain the impact of initial RTO for the case of 10 HTTP sessions, 1000 objects per session, object size of 8 packets, packet size of 1000 bytes, timer granularity of 50ms and 20 background FTP flows. We repeat the test for 13 initial RTO values and run 19 repetitions of each simulation.

Our objective in running this test is simply to confirm that the results of the previous single long run method are reasonably close to the results of another approach with fully randomized test. We did not repeat the completely randomized test for all experiments simply because of time – the experiments of this section alone took a full week of continuous simulation time.

Table 9 Transfer Delay for 19 Repetitions of Test

Average Transfer Delay for each replication for particular initial RTO value												
0.5s	1s	1.5s	2s	2.5s	3s	3.5s	4s	4.5s	5s	5.5s	6s	6.5s
0.8559	0.885	0.8827	0.9063	0.9438	0.9166	1.016	0.993	1.048	1.00152	1.07176	1.1774	1.1376
0.8362	0.8792	0.9633	0.9208	0.9356	0.9471	1.004	1.100	1.00776	1.08052	1.13688	1.1120	1.0724
0.8319	0.8495	0.9072	0.9172	0.9070	1.018	0.9385	1.058	1.023	1.01438	1.11937	1.084	1.1486
0.8432	0.8838	0.9187	0.8748	0.9329	0.9683	0.954	1.030	1.0644	1.04349	1.0845	1.1856	1.0939
0.8319	0.8831	0.8788	0.9051	0.9350	0.9485	1.0260	0.9007	1.0769	1.07684	1.16418	1.0837	1.1212
0.826	0.900	0.9085	0.9352	0.9017	0.9374	0.9622	0.9774	1.0422	1.06826	1.07793	1.0417	1.0764
0.8421	0.8676	0.8875	0.9188	0.9246	0.9514	0.9420	0.9977	1.0356	1.187	1.00381	1.1498	1.028
0.8454	0.8768	0.9140	0.9302	0.9668	0.9361	1.02	0.9983	1.03046	0.9631	1.13166	1.1130	1.0649
0.8411	0.8765	0.8961	0.9115	0.9488	0.9763	0.921	0.9727	1.08934	1.0415	1.10493	1.0384	1.177
0.8324	0.8728	0.9025	0.9028	0.9118	0.9133	1.0245	0.9857	1.0563	1.0815	1.07973	1.1619	1.1576
0.8742	0.9034	0.9011	0.9345	1.01	1.0151	0.9602	1.112	1.06989	1.1072	1.01696	1.1329	1.1277
0.8281	0.9013	0.892	0.9072	0.9504	0.9420	0.9813	0.9703	1.07382	1.0245	1.08762	1.1576	1.0747
0.8376	0.8659	0.909	0.8993	1.008	1.03	0.9632	0.9679	1.085	0.9687	1.13507	1.2955	1.1992
0.8253	0.8520	0.8927	0.9024	0.9446	0.9828	1.0242	0.9982	1.06177	1.0631	1.12281	1.0250	1.1385
0.8234	0.8832	0.8996	0.8906	0.9262	0.9566	0.9226	1.050	1.07715	1.0804	1.07797	1.1119	1.0746
0.8169	0.8969	0.9017	0.9242	0.9216	0.9843	1.0008	1.003	1.05461	1.096	1.10354	1.1029	1.1104
0.8313	0.8693	0.8878	0.8897	0.9516	0.9613	0.9511	1.031	1.03546	1.0401	0.98132	1.2228	1.0945
0.8523	0.8695	0.8686	0.9186	0.9664	0.9878	0.9846	0.9888	1.06795	1.0610	1.10165	1.0640	1.1098
0.8431	0.8907	0.8668	0.9342	0.9444	0.9081	0.9736	0.9852	1.029	1.03191	1.08396	1.14233	1.13408
15.919	16.710	17.0610	17.3246	17.9338	18.2388	18.6887	18.1231	20.030	20.061	20.6667	21.4036	21.1431
0.8378	0.8795	0.89900	0.91181	0.94389	0.96314	0.97893	1.00648	1.05421	1.06891	1.08872	1.12660	1.11279

7.6.3 ANOVA – Results and Analysis

The results of the experiment are provided in Table 9. The table illustrates the average transfer delay for the web objects. For each initial RTO value, the test was repeated 19 times to gain confidence. There were a total of 247 samples thus obtained. The total sum of average transfer delays for a particular level of the factor (particular initial RTO value) is contained in the shaded part of the table. The last line of the table provides the average transfer delay for the 19 replications of that particular level or initial RTO value. Thus, the average for initial RTO value of 1.5s is 0.89s. The overall average for all samples was found to be 0.989215;

Table 10 ANOVA Analysis

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Squares	F
Treatment	$SS_{TREAT} = 1.996$	$k-1 = 12$	$MS_{TREAT} = 0.16$	118.19
Error	$SS_E = 0.3293$	$R-k = 234$	$MS_E = 0.001407$	
Total	$SS_{TOTAL} = 2.325$	$R-1 = 246$		

The ANOVA analysis for the 247 samples are presented in Table 10. The test statistic F is calculated to be 118.19. This is much greater than the critical value for $\alpha=0.05$ and 12 and 234 degrees of freedom – from the F distribution table, F is between 1.75 and 1.83 for the above level of significance and degrees of freedom.

Since the calculated F statistic is greater than the critical value, the test indicates a statistically significant effect due to the factor. The null hypothesis is rejected at the 0.05 level of significance. In addition, the 95% confidence interval for each of the levels (initial RTO values) can be computed as follows:

$$Y_{.j} \pm t_{\alpha/2, R-k} \sqrt{\frac{MS_E}{R_j}}$$

The error term above for the average of each level of the factor is calculated to be ± 0.01695 . In Table 11, we compare the results obtained using the single simulation run method of section 6.4 with the results obtained in this section for each of the 13 initial RTO values tested. The comparison reveals that the single run simulation results are within the confidence interval for all values except initial RTO value of 3.5s and 6.5s where the two results differ beyond the confidence interval error term.

Table 11 Average Delay - Comparing Experimental Methods

Initial RTO Value	Single Run Test (10,100 objects)	19 Repetition Test (1000 objects) - $\bar{Y}_j$
0.5	0.85	0.837876
1	0.88	0.879503
1.5	0.9	0.899001
2	0.92	0.911819
2.5	0.94	0.943893
3	0.96	0.963141
3.5	1	0.978933
4	1.01	1.006481
4.5	1.05	1.054217
5	1.06	1.056911
5.5	1.08	1.088719
6	1.12	1.126501
6.5	1.18	1.112795

Chapter 8 Discussion

In selecting an initial RTO value, there is a tradeoff to be made between two conflicting objectives:

- (i) Reduce the time a flow spends waiting for the timeout
- (ii) Ensure that the flow doesn't undergo needless timeouts due to an initial RTO value much smaller than the RTT.

We refer to the latter objective as the desire to avoid “bad” or “early” timeouts. Some studies [AP99] appear to implicitly give priority to the second objective over the first. We suggest that it is desirable to maximize both objectives to the extent possible – to select an initial RTO value that is larger than most of the RTTs seen on the Internet but is not too large to result in performance degradation for flows with smaller RTTs.

A number of studies [FP98][AH98][HF00][HAM00] report Internet traffic traces where 90% of the RTTs are below 500ms. Given the clear performance impact when the initial RTO is larger than the RTT, we suggest that a value of either 0.5s or 1s be selected as the initial RTO value for TCP flows with the former to be considered if the TCP timer granularity is less than 0.5s.

Changes proposed to the TCP protocol will affect a large deployed base of operating systems. As such, despite the simulation studies showing the performance impact of the initial RTO, there is a clear need to analyze the conditions under which the initial RTO is triggered. This thesis contributes to the work on triggers of initial RTO:

- We identified key loss patterns that cause timeout with initial RTO.
- We determined that these loss patterns occur for one fifth of all lossy flows.

Significant enough presence of the identified loss patterns implies the need for careful selection of initial RTO value. We do recognize that there are specific situations in which loss of SYN, SYNACK will not cause timeout with initial RTO. However, such

situations are not expected to occur very often in near-term real deployed implementations. We consider two such situations below:

- ❖ **Persistent HTTP is utilized** - Thus, for short flows accessing the same web server, instead of starting a new TCP flow per web object on a page, a single TCP connection is used to transfer all objects between a server and the client. In such a case, there is less possibility of timing out with the initial RTO. We note that Joe Touch et al [TH96] showed that persistent HTTP results in little performance improvement for majority of users. Another study by Bernardo et al [HP98] indicates that most users access just one URL at a site before retrieving a URL from another site. This negates the benefit of persistent HTTP.
- ❖ **RTT caching is implemented by the operating systems** – operating systems such as Linux cache the RTT for flows. The cached values are then potentially reused if another TCP flow connects to a host that was previously accessed by this host. Again, while such a scheme reduces the impact of SYN, SYNACK loss, its benefit is limited to cases where the user continually retrieves web pages from the same server. The gain achieved via this scheme is unclear due to the findings mentioned previously in [HP98]. Further, this scheme is not part of the TCP standard and is operating system implementation specific. As such, the TCP standard for initial RTO value cannot rely on the inclusion of this scheme.

Chapter 9 Conclusion and Future Work

This chapter summarizes the conclusions, recommendation and future work arising out of this thesis.

9.1 CONCLUSION

This work evaluates the effect of the initial RTO value on transfer delays for short flows. Analysis showed that a short 10Kbyte HTTP flow utilizing a TCP with large initial RTO value could see its transfer delay increased from 1 second to 6 seconds if the SYN, SYNACK or HTTP GET packets are lost. Examination of Internet traces revealed that SYN/SYNACK constitutes as much as 10% of the total packet loss and 20% of the lossy flows lose their SYN/SYNACK. The above two results provided sufficient motivation to continue with investigation into proper setting of the initial RTO value for TCP senders.

Experiments with TCP implementations in 9 operating systems found that they utilized initial RTO values that range between 3 and 6 seconds. Simulation studies indicated that for loss rates above 5%, TCP timer granularity of 100ms or less, and for RTTs below 500ms, initial RTO values of 3-6s can result in average transfer delays increasing by as much as 70%. Fairness in transfer delays suffered a large proportional decrease in the above cases. Further, we have found that the performance gain by using reduced initial RTO values more than compensates the borderline case of unnecessary retransmissions of SYN and SYNACK due to long RTTs or a receiver who is not present – the performance degradation in terms of reduced average transfer delay is less than 10% when the RTT is as large as 5s.

We recommend that the initial RTO value for TCP be reduced from 3s to 1s. The results also indicate that TCP implementations with timer granularities of 100ms or less and RTTs below 300ms will benefit from an initial RTO value of 0.5s.

9.2 FUTURE WORK

Proposed modifications to the TCP protocol are often subjected to intense scrutiny. New changes are introduced very slowly with only the most conservative of changes being approved for standardization. As such, we recommend further investigation to bolster the results in this work:

- ✓ Investigate possible means of reducing the effects of initial RTO value. Study the effects of obtaining the first RTT sample and updating the RTO on receipt of the SYNACK packet. Particular attention should be given to the case of long-thin networks where the RTT of the SYN packet may be different from the RTT of data packets and may negate benefits of using this method.
- ✓ Perform a large-scale passive analysis of Internet traffic to determine common RTTs on the Internet. All the studies that the author has been able to find to date, measure RTT from one point to a set of points or between a small closed set of points all of which are pre-defined. None of these studies mention studies with modern links. A comprehensive RTT study would use existing traffic traces and need to correlate forward and reverse TCP flows to estimate the RTT for the various flows observed at the monitor points.
- ✓ Study the impact of the initial RTO value on performance when links are faced with extremely high loss rates of 20% or so or cases where each flow has a CWND of one or two packets.
- ✓ Repeat some of experiments in this thesis using real TCP implementations with active studies on the Internet. Some hosts allow configuration of the initial RTO value without requiring kernel changes.

Chapter 10 References

- [Po81] Postel J, "*Transmission Control Protocol*", Internet RFC 793, September 1981
- [Cla82] Clark D, "*Window and Acknowledgement Strategy in TCP*", Internet RFC 813, July 1982
- [BC84] Banks J and Carson J, "*Discrete-Event System Simulation*", Prentice Hall Inc, New Jersey, 1984
- [KP87] Karn P and Partridge C, "*Improving Round-Trip Time Estimates in Reliable Transport Protocols*", Proceedings of ACM SIGCOMM, Stowe, VT, August 1987
- [Ja88] Jacobson V., "*Congestion Avoidance and Control*", In Proceedings of ACM SIGCOMM '88, Stanford, CA, August 1988
- [CJ89] Chiu D and Jain R, "*Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks*", Journal of Computer Networks and ISDN; Volume 17 #1, June 1989
- [Br89] Braden R, "*Requirements for Internet Hosts -- Communication Layers*", Internet RFC 1122, October 1989
- [Jai91] Jain R, "*The Art of Computer Systems Performance Analysis*", John Wiley & Sons, New York, 1991
- [FJ93] Floyd, S., and Jacobson, V., "*Random Early Detection gateways for Congestion Avoidance*", IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p. 397-413.
- [CL94] Comer D and Lin J, "*Probing TCP Implementations*", USENIX, San Francisco, CA, January 1994.

- [St94] Stevens W. R, "*TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP and the UNIX Domain Protocols*", Addison-Wesley, 1994
- [Hoe95] Hoe J, "*Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes*", MSc Thesis, Massachusetts Institute of Technology, June 1995
- [CW95] Claffy K, Braun Werner-Hans, Polyzos G, "*A Parameterizable Methodology for Internet Traffic Flow Profiling*", IEEE JSAC 1995, pp 1481-1494, October 1995
- [TH96] Touch J, Heidemann J and Obraczka K, "*Analysis of HTTP Performance*", USC/Information Sciences Institute, June 1996
- [FF96] Fall K and Floyd S, "*Simulation-based Comparisons of Tahoe, Reno and SACK TCP*", Computer Communication Review, Volume 26, pp 5-21, July 1996
- [Ma97] Mah B, "*An Empirical Model of HTTP Network Traffic*", Proceedings of INFOCOM 97, April 1997, Kobe, Japan
- [Pa97a] Paxson V, "*End-to-End Internet Packet Dynamics*", Proceedings of ACM SIGCOMM, Cannes, France, June 1997
- [Pa97b] Paxson V, "*Automated Packet Trace Analysis of TCP Implementations*", Proceedings of ACM SIGCOMM, Cannes, France, June 1997
- [DJ97] Dawson S, Jahanian F, and Mitton T, "*Experiments on Six Commercial TCP Implementations Using a Software Fault Injection Tool*", In Proceedings of Software Practice and Experience, Vol 1(1); 1997
- [TM97] Thompson K, Miller G, Wilder R, "*Wide-Area Internet Traffic Patterns and Characteristics*", IEEE Network Magazine, November/December 1997
- [BP98] Balakrishnan H, Padmanabhan V, Seshan S, Stemm M and Katz M, "*TCP Behaviour of a Busy Internet Server: Analysis and Improvements*", Proceedings of INFOCOMM 98, San Francisco, March 1998

- [LK98] Lin D and Kung HT, "*TCP Fast Recovery Strategies: Analysis and Improvements*", Proceedings of INFOCOMM 98, San Francisco, March 1998
- [FR98] Feldman A, Rexford J and Caceres R, "*Reducing Overhead in Flow-Switched Networks: An Empirical Study of Web Traffic*", Proceedings of INFOCOMM 98, San Francisco, March 1998
- [Hwa98] Hwang A, "*Observations of Network Traffic Patterns at an End Network: Harvard University*", B.A Thesis, Harvard University, April 1998
- [HP98] Huberman B, Pirolli P, Pitkow J and Lukose R, "*Strong Regularities in World Wide Web Surfing*", Science, April 1998
- [AD98] Aron M and Druschel P, "*TCP: Improving Startup Dynamics by Adaptive Timers and Congestion Control*", Rice University Technical Report #98-318, June 1998
- [AH98] Allman M, Hayes C and Ostermann S, "*An Evaluation of TCP's with Larger Initial Windows*", Computer Communication Review, July 1998
- [CM98] Claffy K, Miller G and Thompson K, "*The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone*", Proceedings of INET, Geneva, Switzerland, July 1998
- [AF98] Allman M, Floyd S and Partridge C, "*Increasing TCP's Initial Window*", Internet RFC 2414, September 1998
- [PN98] Poduri K and Nichols K, "*Simulation Studies of Increased Initial TCP Window Size*", Internet RFC 2415, September 1998
- [All98] Allman M, "*On the Generation and Use of TCP Acknowledgements*", ACM Computer Communication Review, Volume 28, Number 5, October 1998
- [FP98] Fei A, Pei G, Liu R and Lixia Zhang, "*Measurements on Delay and Hop-Count of the Internet*", IEEE GLOBECOM, Sydney, Australia, November 98

- [JV98] Vockler J, "Retransmission Behaviour of Solaris 2.5.1 and 2.6",
<http://www.rvs.uni-hannover.de/people/voeckler/tune/EN/rexmit.html>
- [RF99] Ramakrishnan K and Floyd S, "*A Proposal to add Explicit Congestion Notification (ECN) to IP*", RFC 2481, January 1999
- [PA99] Paxson V, Allman M, Dawson S, Fenner W, Griner J, Heavens I, Lahey K, Semke J, Volz B, "*Known TCP Implementation Problems*", Internet RFC 2525, March 1999
- [AS99] Allman M, Paxson V and Stevens W, "TCP Congestion Control", Internet RFC 2581, April 1999
- [FH99] Floyd S and Henderson T, "*The NewReno Modification to TCP's Fast Recovery Algorithm*", Internet RFC 2582, April 1999
- [AP99] Allman M and Paxson V, "*On Estimating End-to-End Network Path Properties*", Proceedings of ACM SIGCOMM, Cambridge, MA, August 1999
- [FG99] Feldmann A, Gilbert A, Huang P and Willinger W, "*Dynamics of IP Traffic: A Study of the Role of Variability and the impact of Control*", Proceedings of SIGCOMM 99, August 1999, Cambridge, MA
- [KC99] Kulik J, Coulter R, Rockwell D and Partridge C, "*A Simulation Study of Paced TCP*", BBN Technical Report, August 1999
- [Ab99] Abrahamsson H, "*Traffic Measurement and Analysis*", SICS Technical Report, September 1999
- [AF99] Allman M and Falk A, "*On the Effective Evaluation of TCP*", Computer Communication Review, Volume 29 #5, October 1999
- [QZ99] Qiu K, Zhang Y and Keshav S, "*On Individual and Aggregate TCP Performance*", Proceedings of ICNP 99, Toronto, Canada, November 1999

- [LK00] Ludwig R and Katz R, "*The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions*", Computer Communications Review, Volume 30 #1, January 2000
- [BA00] Barakat C, Altman E and Dabbous W, "*On TCP Performance in a Heterogeneous Network: A Survey*", IEEE Communications, January 2000
- [PA00] Paxson, V and Allman M, "*Computing TCP's Retransmission Timer*", Internet Draft, draft-paxson-tcp-rto-01.txt, April 2000
- [GV00] Gorinsky S and Vin H, "*Additive Increase Appears Inferior*", Technical Report 2000-18, University of Texas at Austin, May 2000
- [Hui00] Huitema C, "*Network vs Server Issues in End-to-End Performance*", Workshop on Performance and Architecture of Web Servers (PAWS), Santa Clara, CA, June 2000
- [LS00] Ludwig R and Slower K, "*The Eifel Retransmission Timer*" Computer Communications Review, Volume 30, #3, July 2000
- [HA00] Hadi Salim J and Ahmed U, "*Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks*", Internet RFC 2884, July 2000
- [PF00] Padhye J and Floyd S, "*Identifying the TCP Behaviour of Web Servers*", ACIRI Technical Report, July 2000
- [HF00] Huffaker B, Fomenkov M, Moore D, and Nemeth E, "*Measurements of the Internet Topology in the Asia-Pacific Region*", INET 2000, Yokohama, Japan, July 2000
- [MC00] McCreary S, and Claffy K, "*Trends in Wide-Area IP Traffic Patterns*", ITC Specialist Seminar on IP Traffic Modeling, Measurement and Management, Monterrey, CA, September 2000
- [AG00] Allman M and Griner J, "*TCP Behaviour in Networks with Dynamic Propagation Delay*", Proceedings of IEEE GLOBECOM, San Francisco, CA, November 2000

- [HAM00]** Habib A and Abrams M, "*Analysis of Sources of Latency in Downloading Web Pages*", WEBNET 2000, San Antonio, TX, November 2000
- [NLAa]** National Laboratory for Applied Network Research, <http://moat.nlanr.net/pub/MOAT/Traces>,
- [NS]** NS Simulator, <http://www.isi.edu/nsnam/ns>
- [NLAB]** Press Release, http://dast.nlanr.net/Articles/vbns_extension.html
- [NETP]** Netperf Tool, <http://www.netperf.org>
- [VBNS]** VBNS website, <http://www.vbns.net>