TOWSON UNIVERSITY

OFFICE OF GRADUATE STUDIES

The Performance of Conventional and Bare PC Web Servers under Congestion

By

Alae Loukili

A Dissertation

Presented to the Faculty of

Science

in partial fulfillment of the requirements for the degree

Doctor of Science

in

Information Technology

Department of Computer and Information Sciences

December 2012

TOWSON UNIVERSITY

OFFICE OF GRADUATE STUDIES

DISSERTATION APPROVAL PAGE

This is to certify that the dissertation prepared by ALAE LOUKILI entitled "The Performance of Conventional and Bare PC Web Servers under Congestion" has been approved by the thesis committee as satisfactorily completing the dissertation requirements for the degree Doctor of Science in Information Technology.

[illegible] A. WIJESINHA — 12/7/12
Chair, Dissertation Committee — Date

[illegible] Y. KIM — Dec. 7. 2012
Committee Member — Date

Marius Zimand M. ZIMAND — Dec. 7, 2012
Committee Member — Date

[illegible] R. KARNE — Dec. 7. 2012
Committee Member — Date

Janet V. DeLany — December 12, 2012
Dean of Graduate Studies — Date

ACKNOWLEDGEMENT

I would like to thank my advisor, Dr. Alexander Wijesinha, for his continuous support and advice, and for getting me back to the right path whenever I lost control. He has lifted my spirits the many times I hit the wall. I know sir that our collaboration will continue in the future, and I am looking forward to it. Thank you Dr. Ramesh Karne for giving me the opportunity to be part of history by gaining access to the world of the bare PC, for being available whenever the need arises for some internal debugging in the bare code, and for the many challenges and laughs we had in the lab. To my committee members Dr. Marius Zimand and Dr. Yanggon Kim, I am extremely grateful for your willingness to serve on my committee; your feedback and opinion mean a lot to me and for my work to be judged by you is a great honor.

To “Amigos” in the Bare Machine Computing Lab: Dr. Bharat Rawal, Dr Tony Tsetse, Dr Patrick Appiah-Kubi, Uzo Okafer, and Jeff Liang I say thank you for all the help you granted me during the years of research. To Bill Bullock for his help in improving IPv6 implementation code. To Omar Darwish for his help in resolving formatting issue.

I would like to thank my family for the support and encouragement all this time, and for the many visits they paid us. You have always inspired me to greater heights, and for this, I am highly appreciative.

Finally, an enormous thank you to my lovely wife Aouatif: thank you for your love and support, and for allowing me to take off of my fatherhood responsibilities so I can get my work done.

ABSTRACT

The Performance of Conventional and Bare PC Web Servers under Congestion

By

Alae Loukili

The Transmission Control Protocol TCP has been by far the most used transport protocol in network communication systems. TCP has multiple algorithms to manage its functioning, such as the retransmission timer used to determine how long to wait before the retransmission of any lost or delayed packets occurs, and congestion control mechanisms used to prevent traffic volume from exceeding network resource limits.

We first evaluate the performance under congestion of the popular Windows-based IIS and Linux-based Apache Web servers when serving a single request from a browser. Specifically, we conduct experiments in a real test network with several routers to compare delay, throughput and shared bandwidth percentage when the Web server is subjected to various workloads under different levels of background network traffic. We find that IIS with Compound TCP has performance advantages over Apache with Cubic TCP when the two servers compete for bandwidth, but Apache has smaller delays than IIS for large and medium-sized files.

We next study the performance impact of recently recommended TCP retransmission timer settings using a bare PC Web server with no operating system or kernel running in the machine. We first evaluate server performance in a test LAN with various settings of the alpha and beta constants used for computing SRTT and RTTVAR in the presence of varying levels of background traffic generated by conventional systems. We then study performance with different minimum RTO settings, and compare the performance of the bare PC Web server using the recommended timer settings with the performance of the Apache and IIS Web servers running on Linux and Windows respectively. We find that 1) no combinations of alpha and beta, or sampling strategies, perform consistently better than others under the different levels of background traffic; 2) lower minimum RTO settings than the recommended 1-second minimum will work when there is moderate background traffic, but the 1-second minimum is best when there are higher levels of congestion; and 3) using the standard timer settings but not using the TCP SACK option and congestion control mechanisms degrades bare server performance for some levels of background traffic.

Finally, we conduct experiments using an IPv6 Web server in a test LAN environment with several routers to determine the performance under congestion due to IPv6 and IPv4 traffic. The experiments use an Apache Web server and a bare PC Web server with no operating system. Requests to the servers are made using an ordinary Web browser. Different levels of congestion are created by using MGEN traffic generators. It is found that the IPv4 throughput is slightly greater than (or approximately equal to) the IPv6 throughput under the same level of congestion. When the IPv4 throughput is larger, the differences are between 4-

23%. However, Apache server delays for HTTP requests over IPv6 are between 6-32 ms more than for IPv4 depending on the level of congestion. For all congestion levels, the bare PC Web server has significantly lower throughput and larger delays than Apache regardless of whether IPv6 or IPv4 is used since it does not implement any TCP optimizations. The results show that Web server throughput and delay for browser requests depends on both the congestion traffic rate and the percentage of like traffic in the congestion mix.

This research shows that with respect to TCP congestion control mechanisms on the popular Web servers running on the Windows and Linux operating systems, neither always performs better than the other under various congestion scenarios. So the use of different algorithms together on servers and clients leads to some unfairness when sharing network resources. Furthermore, the choice of an initial value for TCP's retransmission timer, and optimum values for the Alpha and Beta coefficients are difficult to determine as no single pair of values seems to consistently give better performance. Also, Web server performance for requests over IPv6 and IPv4 depend on both the congestion traffic rate and the percentage of like traffic in the congestion mix; and a conventional Web server performs better under congestion than a bare PC Web server due to implementing TCP optimizations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Most applications today use the Transport Control Protocol TCP for communication on LANs and on the Internet. TCP provides reliable data transmission with flow and congestion control. Congestion control algorithms are used to estimate the level of congestion in the network by using loss-based and delay-based techniques. Congestion management is the most complex aspect of TCP; it has been extensively researched and continues to be studied today. Recent research has focused on managing congestion in high-speed long-delay i.e., high bandwidth-delay product (BDP) networks.

The most used TCP congestion control variants are Compound TCP on Windows and Cubic TCP on Linux. Many simulation and real-world studies have confirmed that these two algorithms exhibit good performance and are especially suited for the high BDP networks they target. However, no studies have been done that examine the impact of these two algorithms on the performance of Web servers with browser requests under different workloads, and in the presence of background traffic, by conducting experiments using a real network.

The retransmission timer is a key element of TCP. It is used to set the retransmission timeout (RTO) value that determines when TCP should retransmit. The RTO needs to be set before initiating a connection and subsequently updated based on the smoothed round-trip time (SRTT) and the round-trip time variation (RTTVAR). The latter values are computed using round-trip time (RTT) measurements. The latest IETF recommendation for setting the retransmission timer reduces the initial RTO setting from 3 seconds to 1 second based on the prevalence of today’s high-speed networks and data from recent studies. These studies showed that RTTs of most connections are less than 1 second, retransmission rates during the handshake are about 2%, and about 2.5% of connections have an RTT exceeding 1 second. The IETF recommendations are designed to protect networks from spurious retransmission and congestion collapse.

However, TCP is used in a variety of environments, and by applications that have special needs. Moreover, different algorithms are used for computing SRTT, RTTVAR and RTO (or minimum RTO), and for congestion control by popular operating systems such as Linux and Windows. This has led to alternative approaches to retransmission that have been determined to be safe, but that do not conform to the IETF standard. Clock granularity and the accuracy of RTT measurements (and their frequency) are also of importance when implementing the TCP retransmission algorithm.

The next-generation Internet is expected to have significantly more IPv6 traffic than the Internet does now. Yet, due to the large number of sites including home networks that currently use IPv4 and the many IPv4 address-sensitive applications that would not work over IPv6 without code modification, IPv6 and IPv4 are likely to co-exist for an extended period of time. While IPv4 performance has been researched extensively, fewer studies deal with IPv6 performance, and with performance of servers that handle requests over both IPv6 and IPv4 when the network carries a combination of IPv6 and IPv4 traffic.

This research addresses certain aspects of the preceding issues by studying the performance under congestion of conventional and bare PC Web servers with no operating system. The use of a bare PC Web server enables performance to be evaluated without a conventional TCP/IP protocol stack. It also is convenient for studying the impact of protocol modifications without the need to understand the complex code of a conventional server.

First, we compare the performance of the popular IIS and Apache Web servers that respectively implement Cubic Compound and Cubic TCP congestion control algorithms in a test network with several routers. We send HTTP requests to the Web servers using a Web browser (Internet Explorer or Mozilla Firefox) under varying levels of server load and background traffic to create network congestion. The server workload and background network traffic consists of a mix of HTTP/TCP and UDP traffic generated using the freely-available tools http_load and Mgen respectively. Performance is measured by determining delay, throughput, and shared bandwidth percentage using a Wireshark packet analyzer and http_load. We only compare performance of the default (preconfigured) installations of the two congestion control algorithms (i.e., without making any adjustments to preset parameters). The main finding is that IIS with Compound TCP performs better than Apache with Cubic TCP when the two servers compete for bandwidth, but Apache has lower delays than IIS for large and medium-sized files.

Next, we examine the impact of using the recommended TCP retransmission timer settings. Our studies use a bare PC Web server [2] in which the server application is intertwined with lean implementations of the necessary network protocols and there is no operating system or kernel running in the machine. The experiments are conducted in a small test LAN with several routers. We first measure bare PC Web server performance with various settings of the alpha and beta constants used for computing SRTT and RTTVAR in the IETF algorithm, and with different RTT sampling strategies (frequencies). These measurements are done in the presence of different levels of background traffic generated by conventional systems to create congestion. We then examine the effect on retransmission of using alternative minimum RTO settings. Finally, we compare the performance of the bare PC Web server using the recommended settings with the performance of the Apache and IIS Web servers running on Linux and Windows respectively.

We find that no particular combinations of alpha and beta, or sampling strategies, perform consistently better than others under the different levels of background traffic. Also, much lower minimum RTO settings than the recommended 1-second minimum will work when there is only moderate background traffic, but the 1-second RTO minimum is best when there are higher levels of congestion. However, not using congestion control even with the standard timer settings degrades performance depending on the level of background traffic.

Finally, we evaluate the performance of the Apache and bare PC Web servers under congestion resulting from different mixes of IPv6 and IPv4 traffic by measuring network throughput and delay. The experiments are conducted in a test LAN environment consisting of several subnets connected by routers. MGEN (MultiGenerator) traffic generators are used to create IPv4 and IPv6 background traffic at moderate and higher levels of network congestion. We use primarily TCP background traffic to reflect its predominant use in the Internet and a small amount of background UDP traffic to represent applications such as VoIP, live video, or support protocols such as DNS or DHCP. The use of a bare PC Web server with no operating system enables the impact of operating system overhead, and of not using TCP optimizations or congestion control, to be determined. The throughput and delay measurements are obtained by making requests to the Web servers using an ordinary (Firefox) Web browser. The main findings are that 1) Web server throughput is not significantly different for requests over IPv6 and IPv4; 2) Delays over IPv6 can be much larger than delays over IPv4; 3) the Apache Web server performs significantly better than the bare PC Web server for all levels of congestion; and 4) throughput and delay depend on both the congestion traffic rate and the percentage of like traffic causing the congestion.

The rest of this dissertation is organized as follows. In Chapter 2, we provide background information on Computing Systems, Cubic and Compound TCP, TCP’s retransmission timer and congestion in IPv6 networks. We also discuss related work in this chapter. In Chapter 3, we experiment with Cubic and Compound TCP, and compare their performance. In Chapter 4, we examine the impact of changing various parameter settings in the bare PC Web server’s TCP’s retransmission timer and compare results with default settings on the Apache and IIS Web servers. In Chapter 5, we study an IPv6 Web server under different levels of network congestion. In Chapter 6, we summarize our contributions and present the conclusion. The Appendix consists of tables containing the data tables from our performance studies used to obtain the results discussed in Chapters 3-5; it is separated by chapter. The bibliography contains the references used throughout this work.

CHAPTER 2

RELATED WORK

2.1 Computing Systems

2.1.1 Conventional systems

Conventional systems that run on an operating system (OS) such as Linux and Windows provide a traditional TCP/IP protocol stack that is part of the OS kernel, and a socket interface that applications can use. The popular Web servers Apache and IIS use the standard TCP/IP stack provided by the respective operating systems Linux and Windows.

2.1.2 Bare Machine systems

Bare PC systems are based on the Dispersed Operating System Computing (DOSC) concept introduced in [30]. Bare C++ interfaces to the hardware are described in [31]. In Bare PC or Bare Machine Computing (BMC) systems, applications including servers and clients can run directly on the hardware without using any OS or kernel. The protocol design in a bare PC for any application, and in particular, a Web server, is not bound by strict network layering conventions [14]. The TCP and HTTP protocols are intertwined, and the bare machine system uses a lean version of TCP, lacking for instance any congestion algorithms.

2.2 Cubic and Compound TCP

Numerous studies have been conducted on the congestion control aspects of TCP. For example in [1], the TCP congestion window update algorithm is modified to address the issue of RTT-fairness, and it is shown by a combination of analysis, simulation and testbed studies that protocol stability and efficiency are maintained. In [2], simulation studies are used to show that an existing TCP variant is suited for use on the local segment of a spliced connection to address the needs of wireless multimedia flows with home entertainment centers. While many variants of the basic TCP "Reno" congestion control algorithm [3] have been proposed, relatively few have seen large-scale deployment in real-world networks.

Two notable exceptions are Compound TCP [4, 5] used by the Windows operating system and Cubic TCP [6] used by the Linux operating system that attempt to address the needs of today's high BDP networks. Compound TCP (also known as C-TCP) modifies the basic congestion control algorithm by adding a scalable term to the congestion control window based on the standard "Vegas" estimate of network buffering. In contrast, Cubic TCP models the congestion window as a cubic function of the time since the last detected loss that also depends on the congestion window size prior to that loss. While both variants are known to perform well under congestion in high BDP networks and have good fairness properties, Compound TCP may not treat all flows equally and Cubic TCP may have sub-optimal network utilization and increase packet loss [7].

Our study examines performance of the IIS and Apache Web servers with their respective Compound and Cubic TCP variants under different workloads for a single browser request in a network when there are competing flows and other background traffic. Most TCP studies of its congestion control variants use simulation. For example, [8] and [9] study the performance of the two variants using a simulation tool. However, [10] suggests that simulation studies may not reflect the complexity of TCP protocol behavior with traffic in a real network. Also, in [11] it is observed that real implementations of congestion control algorithms often make changes to parts of the TCP stack that are not directly related to the congestion algorithm to improve overall performance. We consider "out-of-the-box" (i.e., default) implementations of TCP in the two most popular Web servers and their operating systems running on real hardware. Finally, we use background traffic in accordance with the guidelines in [12] to study performance of the two TCP variants.

2.3 TCP’s retransmission timer and the minimum RTO

Many studies have examined various aspects of TCP’s retransmission timer algorithm. In [15], the authors determine how initial RTO settings (used by various operating system versions at the time) impact packet loss rate and suggest changing the initial RTO from 3 seconds to between 500 ms and 1 second. In [16], the RTO algorithm is enhanced by making it window-based rather than round-trip-time-based so that unnecessary retransmissions and unfair resource allocation could be minimized. In [17], the same authors argue that a 1-second minimum RTO is only needed because of spurious retransmissions that are due to clock granularity and delayed acks. They note that the first problem is not an issue in modern operating systems, and then provide a mechanism to extend the minimum RTO to deal with the second problem. The mechanism is shown by simulation studies to improve TCP performance in the case of a wireless link connected to a high-speed backbone.

The Eifel response algorithm allows a TCP sender to prevent the negative effects of a detected spurious timeout by adapting the retransmission timer [18]. In [19], high-resolution timers are used to solve the problem of throughput collapse in datacenter Ethernets due to TCP timeouts resulting from synchronized request workloads. In [20], a large number of real TCP traces are analyzed, and it is found that RTT values within each connection vary widely. In [21], a new algorithm F-RTO for recovery after retransmission timeout is proposed, and it is shown that unnecessary retransmissions can be avoided without using TCP options. In contrast to these previous studies, we implement the RTO algorithm on a bare PC Web server using the recently recommended TCP timer settings from [13], and study its impact on RTT values and on the performance of the server.

2.4 Congestion in IPv6

A recent study on IPv6 performance [22] concludes that RTTs for IPv6 connections are less than for IPv4, although they have higher packet loss. Higher loss over IPv6 is also noted in [23], although they find that delays over IPv6 are larger, which is also supported by studies of real-time voice and video in [24]. In [25], it is claimed that native IPv6 has significantly better throughput than IPv4 due to enhanced routing capabilities. In contrast, based on measurement studies, it is found in [26] that routing inefficiencies are the cause of poor IPv6 performance although IPv4 and IPv6 performance for data are compatible. In [6], using values of throughput, delay and other metrics in a testbed, it is determined that network performance for IPv4 and IPv6 may differ depending on traffic types and the operating system.

Global Internet measurements are used to compare latency and loss over IPv4 and IPv6 in [28]. While overall performance over IPv4 is often better, about 10% of the time, latency over IPv6 can be between 10-38 ms less. Internet packet traces are used in [29] to study features of IPv6 packets. It is shown that IPv6 traffic has more self-similarity than IPv4 traffic resulting in poorer performance.

Extensive studies on network congestion have been conducted. A survey of TCP congestion control methods is given in [7]. While the IP version does not affect TCP congestion control directly, differences in header sizes and packet processing delays can impact network performance under congestion.

We evaluate the performance of Apache and bare PC Web servers under congestion resulting from different mixes of IPv6 and IPv4 traffic by measuring network throughput and delay. Our study differs from the above studies since they do not specifically determine the throughput and delay associated with browser requests over IPv6 and IPv4 to a Web server under a mix of IPv6 and IPv4 congestion traffic. We also use an IPv6-IPv4 capable bare PC Web server with no operating system. The implementation and performance of a bare PC Web server that runs over IPv4 are described in [14]. The IPv4-IPv6 capable bare PC Web server used in this study was built by modifying an IPv4 bare PC Web server.

CHAPTER 3

Web Server Performance With Cubic And Compound TCP

3.1 Experimental Setup

3.2.1 Test Network and Traffic Generation

For our experiments, we used a network of clients connected to a network of servers via four Linux routers R1-R4 and five Ethernet switches S0-S4 as shown in Figure 1.

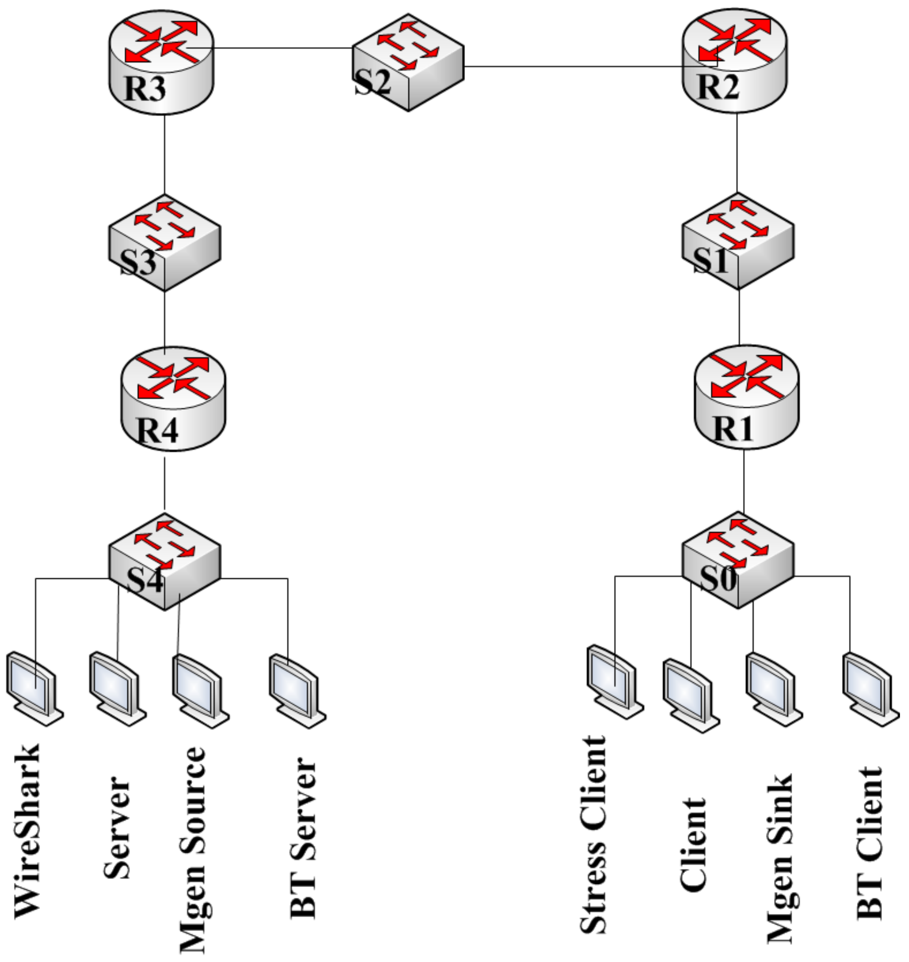

Figure 1. Test Network1

Details of the hardware and software used for the experiments are given in Table 1.

TABLE 1. HARDWARE AND SOFTWARE SPECIFICATION

Device/Function	**Details**
Switch S0	Netgear/FSM726S
Switch S1	Cisco/SG100d-08
Switch S2	Linksys/EG005W
Switch S3	Cisco/SG100d-08
Switch S4	Netgear/GS108T
Packet Analyzer	Dell OPTEPLEX GX520;XP/ Wireshark Version 1.2.7 (SVN Rev 32341)
Servers	Dell OPTEPLEX GX520; Windows server 2008 (IIS 7); Fedora 12 (Constantine) Kernel Linux 2.6.31.5-127.fc12.i686 Apache HTTP Server 2.2.16
Mgen source	Dell OPTEPLEX GX260; Windows XP Professional 2002 SP3
Mgen sink	Dell OPTEPLEX GX260; CentOS Version 2.16.0
Clients	Dell OPTEPLEX GX520; Windows XP Professional 2002 SP3; Linux Fedora 12 Kernel 2.6.31.5-127.fc12.i686; browser: Firefox v3.6.7, Internet Explorer 8.0.6001.18702
Routers	Dell OPTEPLEX GX520; Fedora 12 Kernel Linux 2.6.31.5-127.fc12.i686

All NICs and switches are 1 Gbps except for R1's NIC on its internal (i.e., client network side) interface, and switch S0, that are 100 Mbps in order to cause congestion. In the absence of other traffic, the RTT between routers R1 and R4 is approximately 0.3 ms.

The Web browser on the client (referred to as "Client" in the figure) sent a single request for a 504, 320, or 160 KB file (to represent large, medium and small files respectively) to the Web server (referred to as "Server" in the figure) in the presence of background traffic on the network and stress traffic to the Web server "Server" as described below. The experiments were done using Mozilla Firefox and repeated using Internet Explorer as the browser on "Client". Since the results with both browsers were similar, only the results using Firefox are reported.

The congestion scenarios for the experiments (described below) were created using the test network in Fig. 1 (or minor variations of it) by generating traffic as follows. For background traffic, a mix of UDP and HTTP/TCP traffic was generated using the Mgen [32] and http_load [33] generators respectively: the "Mgen Source" sent UDP packets to the "Mgen Sink" at varying rates, and a background traffic client (referred to as "BT Client") sent multiple HTTP requests at varying rates via http_load for a 320 KB file to a background traffic Web server (referred to as "BT Server"). In addition, stress traffic in the form of competing requests to the Web server "Server" at varying rates for a 320 KB file was generated by a client (referred to as "Stress Client") using http_load. The combination of background and stress traffic creates a traffic mix for examining the behavior of the two congestion control variants.

3.2.2 Congestion Secanarios

We studied performance of the Apache/Linux and IIS/Windows Web servers with their respective TCP congestion control approaches (Cubic TCP and Compound TCP) by sending browser requests to the servers under three congestion scenarios C1, C2, and C3. The scenarios create different mixes of background and stress traffic on the network totaling 100 Mbps in each case as described below.

C1: Background traffic is 10 Mbps of UDP traffic from Mgen Source to Mgen Sink, and 40 Mbps of HTTP/TCP traffic generated by requests from BT Client to BT Server i.e., the background traffic is 20% UDP and 80% TCP. The Web server "Server" additionally has a workload of 50 Mbps of HTTP/TCP stress traffic due to requests from Stress Client.

C2: Background traffic is 15 Mbps of UDP traffic from Mgen Source to Mgen Sink, and 60 Mbps of HTTP/TCP traffic generated by requests from BT Client to BT Server i.e., the background traffic is 20% UDP and 80% TCP. The Web server "Server" additionally has a workload of 25 Mbps of HTTP/TCP stress traffic due to requests from Stress Client.

C3: Background traffic is 10 Mbps of UDP traffic from Mgen Source to Mgen Sink, and 30 Mbps of HTTP/TCP traffic generated by requests from BT Client to BT Server i.e., the background traffic is 25% UDP and 75% TCP. The Web server "Server" additionally has a workload of 60 Mbps of HTTP/TCP stress traffic due to requests from Stress Client.

The differences between the scenarios can be characterized as follows: for UDP background traffic, scenarios C3 and C1 have the same levels, while C2 has 5 Mbps more. For TCP background traffic, C2 has 20 Mbps more than C1, which has 10 Mbps more than C3. For TCP stress traffic, C3 has 10 Mbps more than C1, which has 25 Mbps more than C2. Comparing TCP stress traffic and the total (TCP and UDP) background traffic, C1 has the same levels, while C3 has 60% stress and 40% background, and C2 has 75% background and 25% stress.

For all three scenarios, we start Mgen, run http_load on Stress Client and BT Client, and send a single request from the Web browser on "Client" to the Web server on "Server". The

Wireshark packet analyzer [35] is used on the server side to measure delay and throughput. Information from http_load running on the Stress and BT Clients is used to compute throughput on the client side. Measuring the throughput on both the client and server side enables the impact of congestion control with and without network delays to be studied.

3.2 Experimental Results

3.2.1 Same Server Type

In the first set of experiments, Server and BT Server are the same type i.e., both are Apache or both are IIS. Traffic is generated according to the congestion scenarios C1, C2 and C3. Figs. 2-4 show the delay for a single browser request, and Figs. 5-7 show the server throughput using large, medium and small files respectively.

The server delay includes the processing time and the delays introduced by the congestion control algorithm. The former depends on the total workload (the file size, the total number of bytes processed, and the OS/server implementation), and the latter depends on the congestion level and the characteristics of the algorithm. Observed delays may be due to a combination of factors. For example, a more aggressive backoff policy to address increased congestion is compensated for by smaller delays due to the reduction in traffic and packet loss.

Let t_server(C) denote the total server delay in responding to the browser request for a scenario C. Fig. 2 shows that for large files, t_IIS(C3) > t_IIS(C1) ≈ t_IIS(C2) ≈ t_Apache(C2) > t_Apache(C3) > t_Apache(C1). For IIS, when serving a large file, an increase in the workload due to stress traffic increases the delay as expected in scenario C3 over scenario C1.

However, comparing scenario C1 with C2, the stress traffic is reduced by 50% in scenario C2, and the background traffic is increased by 50%, but there is no difference in the delay. This appears to suggest that Compound TCP has reduced the transmission rate due to congestion resulting in a delay that is the same as the delay for C1. For Apache, the relative performance for scenarios C1 and C3 is the same as for IIS, but the congestion due to the increased background traffic in C2 has resulted in larger delay than in C1. This means that Cubic TCP reduces its transmission rate more aggressively than IIS when congestion increases, resulting in a greater delay to serve a large file.

For medium files, Fig. 3 shows that t_IIS(C3) ≈ t_IIS(C1) > t_IIS(C2) > t_Apache(C3) ≈ t_Apache(C1) > t_Apache(C2). In this case, the additional stress traffic in C3 compared to C1 did not increase the delay because the requested file size is smaller and because both congestion control variants have reduced the transmission rate in C1 due to the increased background traffic. In the case of scenario C2, the reduced file size and the highest level of background traffic result in the least delay because each congestion variant has reduced its transmission rate sufficiently.

For small files, Fig. 4 shows that t_IIS(C1) < t_Apache(C1) ≈ t_IIS(C3) < t_IIS(C2) < t_Apache(C2) < t_Apache(C3). Here, the delays for scenarios C1 are less than for C3 as expected since it has less stress traffic, but the delay for C2 is larger than for C1 with both servers. Since the file size is small, the increased delays for C2 are due to reducing the transmission rate because of the congestion caused by the additional background traffic. In this case, Cubic TCP has a larger delay than Compound TCP because it is reducing its transmission rate more aggressively.

In general, the above results indicate that the Apache server is more efficient than IIS for large and medium files, but less efficient for small files. They also indicate that the Apache server has reduced performance compared to IIS when congestion increases as in scenario C2. This is because Cubic TCP adopts a more conservative approach than Compound TCP when losses are detected. More studies with a variety of congestion scenarios are needed to validate this claim.

In Figs. 5-7, NRT refers to the server throughput when retransmissions are ignored. It is seen that scenario C2 has the lowest and scenario C3 has the highest server throughput regardless of file size as would be expected. However, throughput for IIS is slightly higher than the corresponding throughput for Apache in all cases. This suggests that Compound TCP (IIS/Windows) may be transmitting more aggressively under congestion than Cubic TCP (Apache/Linux) since the throughput is higher even for large and moderate file sizes (when delays for IIS were higher). The throughput in each scenario for all file sizes is about the same because the stress traffic level in each scenario is the same, and the difference between the small, medium and large file sizes used for the single request made by the browser does not affect the throughput.

Fig. 8 shows the throughput measured at the Stress and BT clients with a request for a 320 KB file. As with server throughput, client throughput is also almost identical for the three file sizes, so only the results for one file size are shown here. While client throughput is less than server throughput due to network delay, relative relationships between throughput values are not preserved. For example in scenario C1, IIS and Apache throughput measured at the client is about the same, whereas IIS NRT throughput measured at the server side is slightly higher than the Apache NRT throughput (Fig. 6). This difference could be insignificant or due to a slightly lower packet loss with IIS (and hence fewer retransmissions by the IIS server). More detailed studies are needed to understand the relation between Compound and Cubic TCP, and the differences in observed throughput at the client and the server.

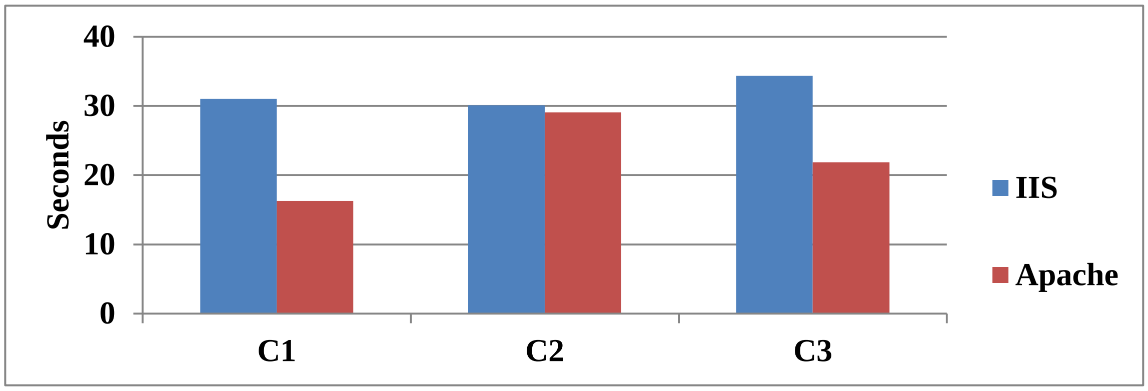

Figure 2. Server delay: 504 KB

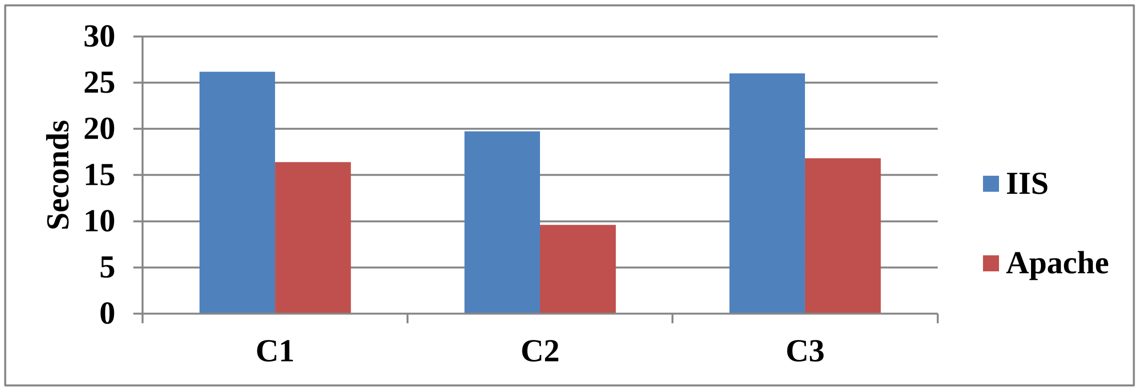

Figure 3. Server delay: 320 KB

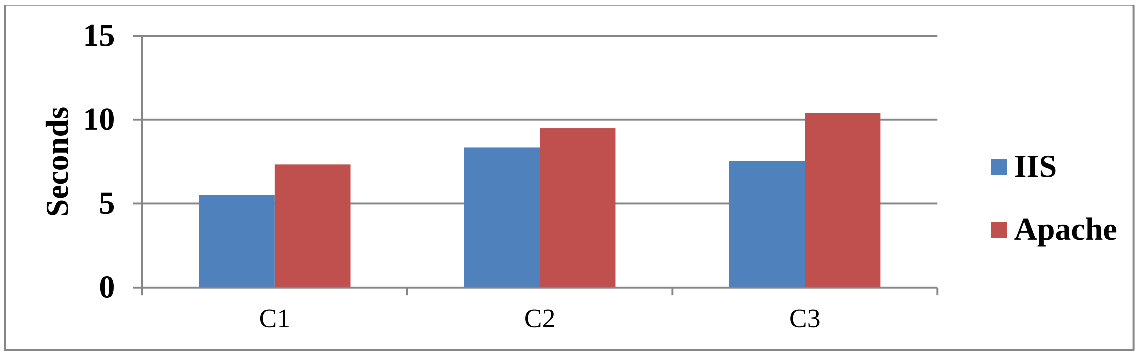

Figure 4. Server delay: 160 KB

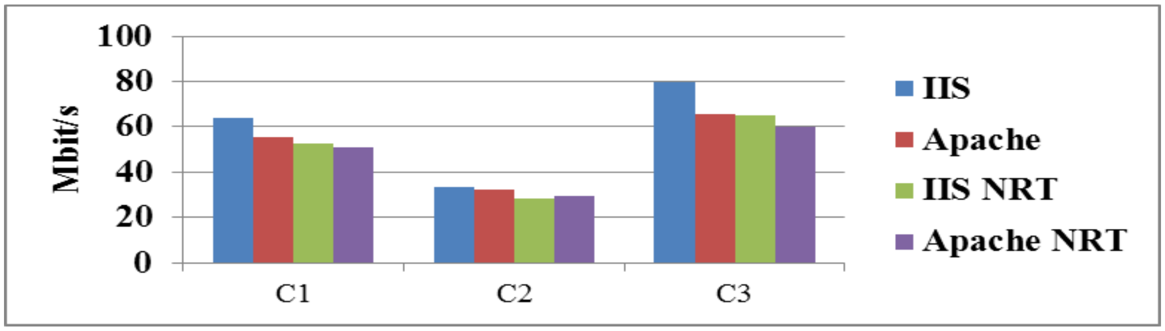

Figure 5. Server throughput: 504 KB

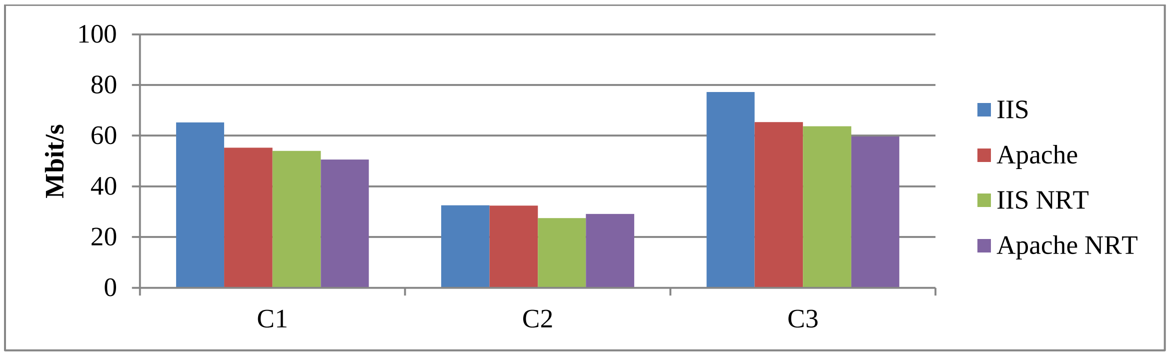

Figure 6. Server throughput: 320 KB

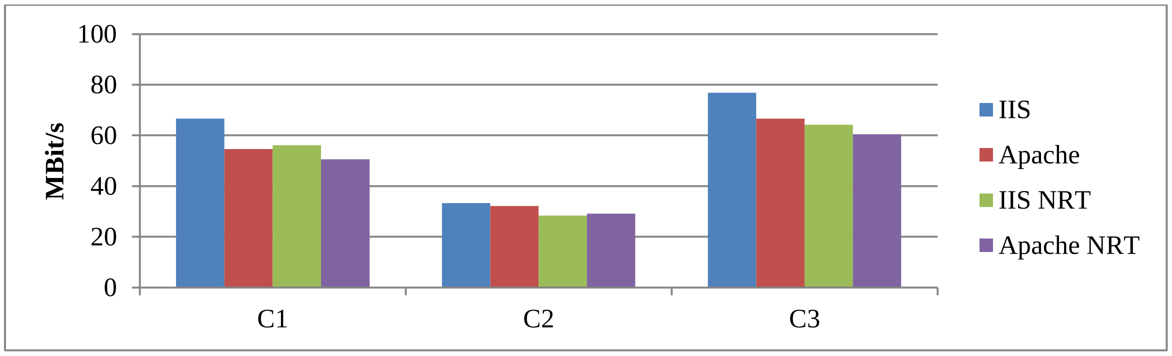

Figure 7. Server throughput: 160 KB

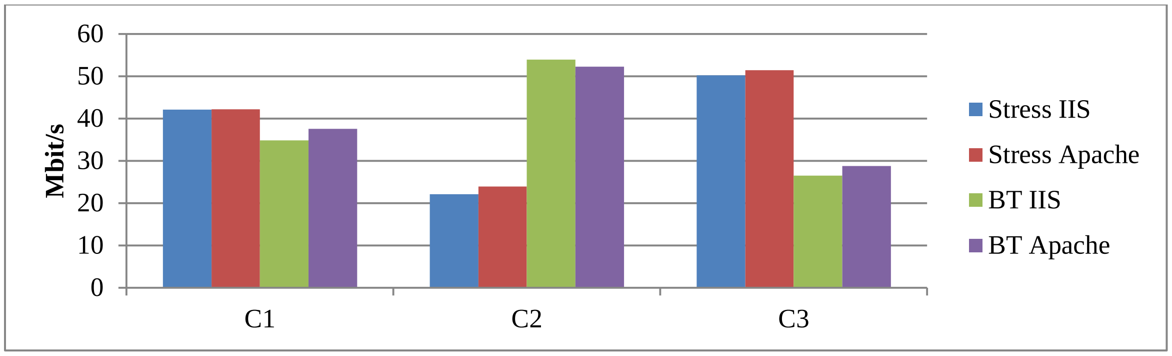

Figure 8. Client throughput: 320 KB

Table 2 shows the bandwidth percentage shared between the Stress Traffic (ST) and Background Traffic (BT) servers. For IIS, each server gets roughly an equal share of the bandwidth, whereas for Apache, one of the two servers is always getting a larger share. For instance in scenario C1, the maximum possible rates are 50 Mbps of TCP stress traffic and 40 Mbps of TCP background traffic. For IIS, 84.1% of the maximum stress traffic rate and 87% of the maximum background traffic rate are achieved, which is a 2.9% difference. For Apache, 84.4% of the maximum stress traffic rate and 93.9% of the maximum background traffic are achieved, which is a 9.5% difference. This result shows that Compound TCP enables better sharing of the available bandwidth with another like flow (i.e., Compound TCP) than Cubic TCP.

TABLE 2. SHARED BANDWIDTH PERCENTAGE (SAME SERVER TYPE)

Scenario	**ST IIS**	**BT IIS**	**ST Apache**	**BT Apache**
C1	84.1	87.0	84.4	93.9
C2	88.5	89.9	95.7	87.0
C3	83.6	88.3	85.7	95.9

3.2.2 Different Server Type

In the second set of experiments, a file of the same size (504 KB) is requested by the browser under all three congestion scenarios. For these experiments, Server and BT Server are of different types (i.e., Server is IIS and BT Server is Apache, or Server is Apache and BT Server is IIS).

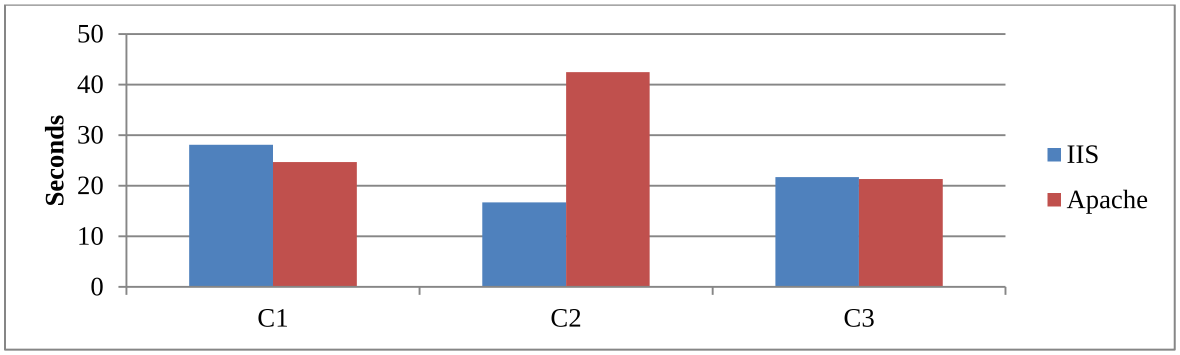

Figure 9. Delay (different servers): 504 KB

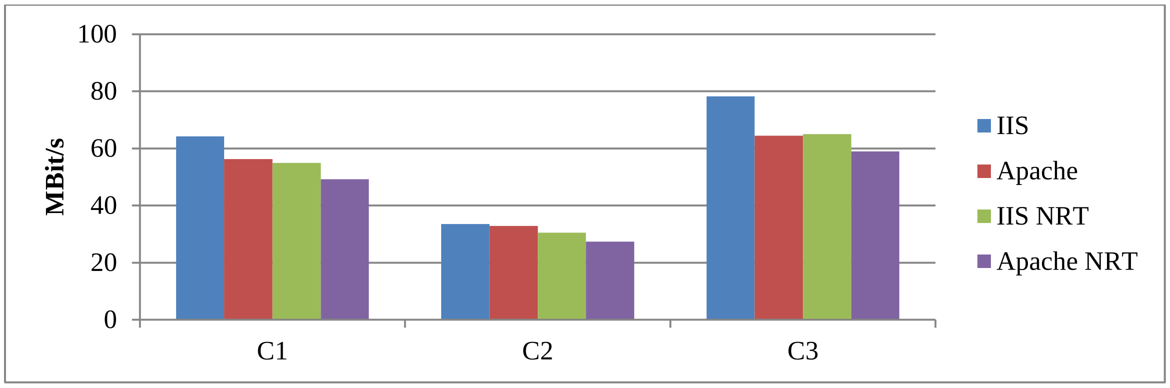

Figure 10. Server throughput (different servers): 504 KB

In Fig. 9, it is seen that for scenario C2 (in which background traffic is at its highest level), the delay for Apache when serving a single browser request is 3 times that for IIS when the stress and background servers are of different types. In contrast, for scenarios C1 and C3, the

IIS delay is only slightly larger than the Apache delay. Comparing with the case when the servers were of the same type (Fig. 2), the IIS delays for C1 and C3 are larger than for Apache, but the difference between IIS and Apache for C2 is insignificant. It therefore appears that the reduction in the transmission rate under congestion is larger for Cubic TCP (Apache) than it is for Compound TCP (IIS).

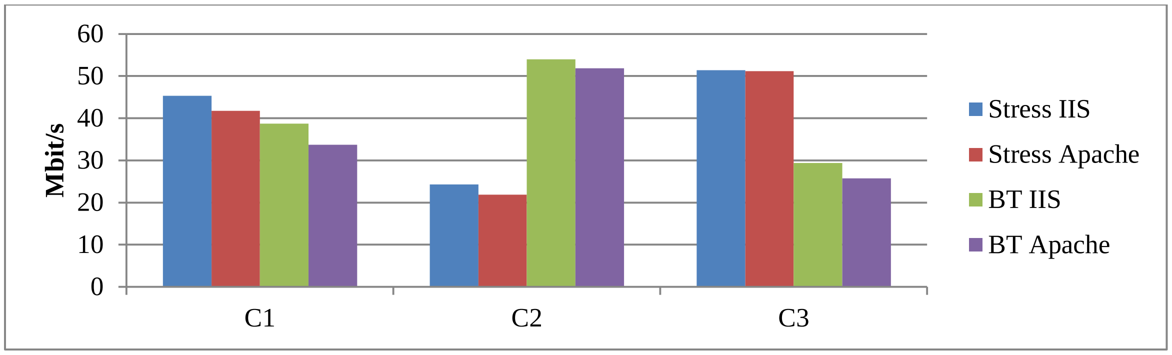

Figure 11. Client throughput (different servers): 504 KB

In Figs. 10 and 11, server and client NRT throughput with different server types is similar to the result with servers of the same type (Figs. 5 and 8). The IIS server now has slightly larger throughput than Apache for all scenarios.

TABLE 3. SHARED BANDWIDTH PERCENTAGE (DIFFERENT SERVERS)

Scenario	**ST IIS**	**BT Apache**	**ST Apache**	**BT IIS**
C1	90.7	84.3	83.5	96.8
C2	97.1	86.4	87.4	90.0
C3	85.7	85.8	85.2	98.0

Table 3 shows the bandwidth percentage shared between the Stress Traffic (ST) and Background Traffic (BT) servers when the servers are of different types. In scenario C1, IIS achieves 90.7% of the maximum stress traffic rate and Apache achieves 84.3% of the maximum background traffic rate, which is a 6.4% difference. When the servers are switched, IIS achieves 96.8% of the maximum background traffic rate and Apache achieves 83.5% of the maximum stress traffic rate, which is a 13.3% difference. Similarly in scenario C2, IIS achieves 97.1% of the maximum stress traffic rate and Apache achieves 86.4% of the maximum background traffic rate; and in scenario C3, IIS achieves 98.0% of the maximum background traffic rate and Apache achieves 85.2% of the maximum stress traffic rate. This shows that IIS can achieve most of its allowed maximum traffic rate when sharing the network with an unlike flow (i.e., Cubic TCP), which can only achieve about 84-86% of its allowed maximum traffic rate. Even when the shared percentages between Compound and Cubic TCP are more equitable, Cubic TCP did not achieve more than 87.4% of its allowed rate. Thus, Cubic TCP defers to Compound TCP in a congested network, which limits the performance of the Apache Web server.

3.2.3 Competing Servers

In the third set of experiments, we modify the test network to include two Web servers Server1 (S1) and Server2 (S2), two clients Client1 and Client2, and two Stress clients C1 Stress and C2 Stress. In addition to 10 Mbps of UDP traffic from Mgen, 30 Mbps of background HTTP/TCP traffic are generated by requests from BT Client (Cbt) to BT Server (BTS), and each server S1 and S2 has a workload of 30 Mbps of HTTP/TCP stress traffic due to requests from the clients C1 Stress and C2 Stress respectively.

As in the earlier experiments, the combined stress traffic and background traffic total 100 Mbps. The browser on client Client1 then sends a single request to the Web server S1, and simultaneously the browser on client Client2 sends a single request to the Web server S2. Each request is for a 504 KB file.

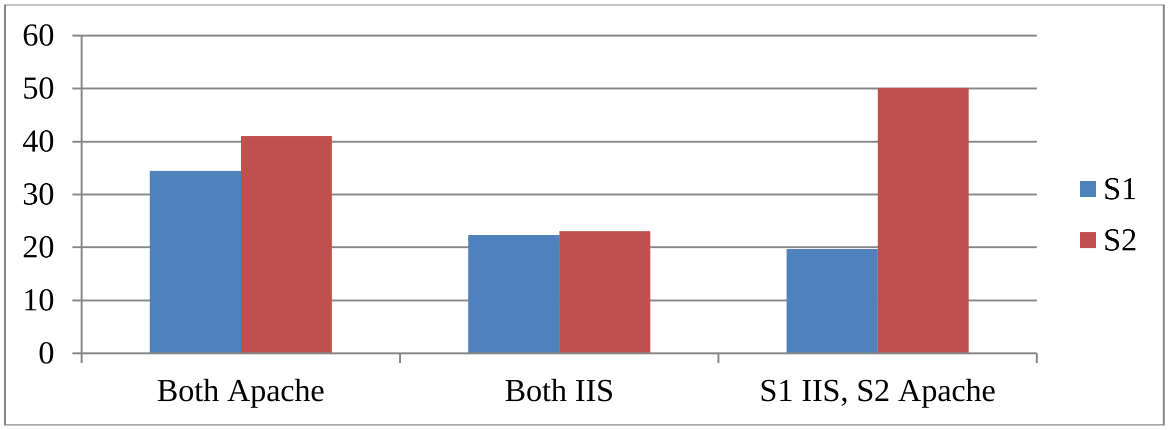

Figure 12. Server delay (competing servers): 504 KB

Fig. 12 compares the server delay when 1) both S1 and S2 run Apache, and BTS is IIS; or 2) both S1 and S2 run IIS, and BTS is Apache; or 3) S1 runs IIS and S2 runs Apache, and BTS is Apache. In 1) and 2), the delays are comparable. Although the two Apache servers do not share the bandwidth equally and experience delays that differ by about 5 seconds (i.e., 35 and 40 second delays), the IIS servers have equal delays of about 20 seconds. In 3), the performance of Apache degrades to an unacceptable level (a delay of 50 seconds), whereas IIS maintains its 20 second delay. It is clear that IIS (Compound TCP) performs better than Apache (Cubic TCP) in this case.

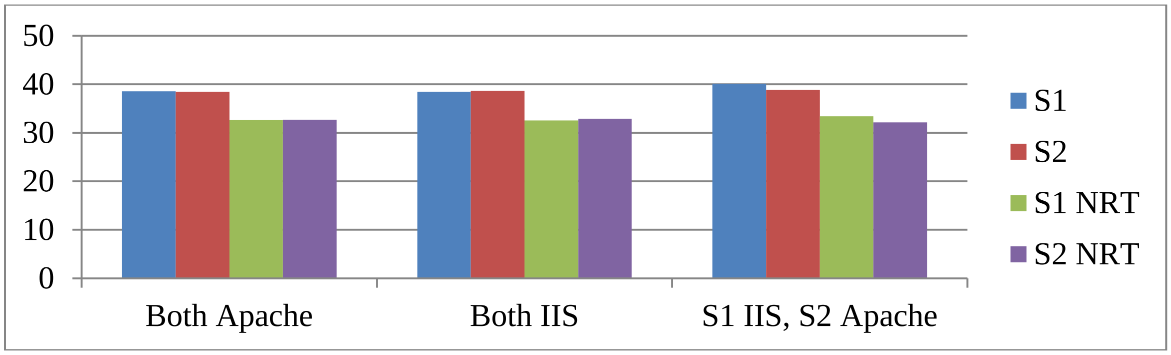

Figure 13. Server throughput (competing servers):504 KB

Fig. 13 shows the server throughput for this experiment. It can be seen that the throughput values for both servers are comparable in each of the three cases. Fig. 14 compares the throughput values measured at the client side. Again, the throughput values are comparable, although the throughput at the background client is marginally less than the throughput at each of the two stress clients.

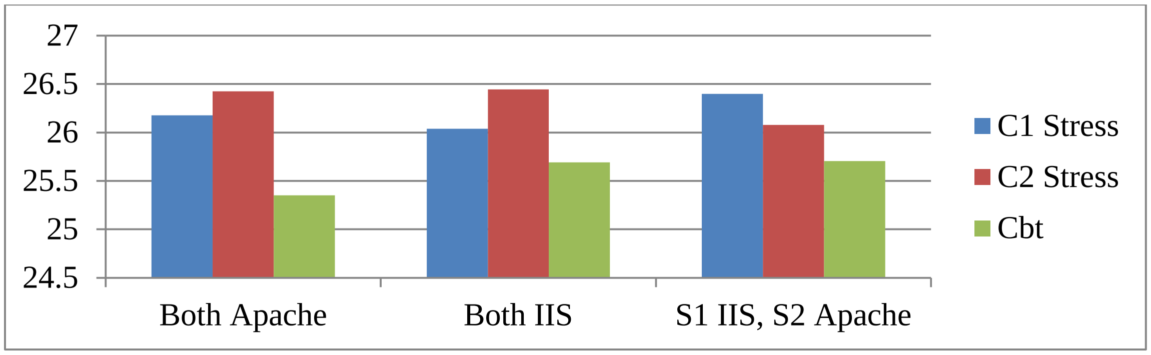

Figure 14. Client throughput (competing servers): 504 KB

Table 4 shows the bandwidth percentage shared between the three flows measured at the client side for the case of competing servers. It can be seen that each flow gets an approximately equal percentage of the maximum allowed bandwidth.

TABLE 4. CLIENT THROUGHPUT (COMPETING SERVERS): 504 KB

Scenario	**C1 Stress**	**C2 Stress**	**Cbt**
S1, S2 Apache, BTS IIS	87.3	88.1	84.5
S1, S2 IIS, BTS Apache	86.8	88.1	85.6
S1 IIS, S2 Apache, BTS Apache	88	86.9	85.7

CHAPTER 4

TCP'S RETRANSMISSION TIMER AND THE MINIMUM RTO

4.1 Experiments and results

4.1.1 Setup

The experiments use background traffic consisting of 10% UDP and 90% TCP at three different levels: 75 Mbps, 100 Mbps and 125 Mbps. The background traffic is generated as follows: BT client uses http-load [10] to generate HTTP/TCP traffic by requesting a 320-KB file from BT Server (Apache HTTP Server 2.2.16 running Fedora 12 (Constantine) Kernel Linux 2.6.31.5-127.fc12.i686), while Mgen source uses the mgen traffic generator [32] to generate UDP traffic to the Mgen sink. Packets are captured using the Wireshark analyzer [34] at the server side. The traces are used to deduce the values of throughput and delay when the Web browser (Internet Explorer 8.0.6001.18702 with Windows XP Professional 2002 SP3) on the client "Client" requests a 320-KB file from the Web server "Server" (by computing the time between the "GET" request and the last valid ack sent by the client). Results using the Firefox v3.6.7 browser with Linux Fedora 12 Kernel 2.6.31.5-127.fc12.i686 were similar. To determine the effect of changing the values of alpha and beta, the bare PC Web server TCP code was modified accordingly. The measured sample values of RTT were then used to compute the smoothed round trip time (SRTT), the variance in round trip time (RTTVAR), and retransmission timeout (RTO) as follows using the standard algorithm [13]:

RTTVAR= (1-beta)*RTTVAR +beta* |SRTT – RTT|

SRTT = (1- alpha)*SRTT + alpha*RTT

RTO = SRTT + max (G, K*RTTVAR)

Here K=4, and G is the clock granularity, which is 250 microseconds for the bare PC. We examined the effect of changing the values of alpha and beta, and also the effect of computing the value of RTO using 3 different sampling strategies: 1) All: where the RTT measurement is based on the RTTs of all acknowledged packets per window of data; 2) One: where the RTT measurement is based on the RTT of one acknowledged packet per window of data; 3) Some: where the RTT measurement is based on the RTT of some acknowledged packets per window of data (typically, alternate acknowledgements were used). The alpha and beta pairs whose combinations were used as the nine weights W1-W9 are shown in Table I. The weight W1 uses the values of alpha and beta from [13]. This weight together with the sampling strategy "One" (one sample per RTT) is commonly used in TCP implementations.

4.1.2 Throughput and Delay

For each weight W1-W9 and each of the three sampling strategies "All", "One", and "Some" defined above, the throughput and delay with respect to the 320-KB file were measured for background traffic levels of 75, 100, and 125 Mbps. The results are shown in Figs. 15-20.

TABLE 5. ALPHA AND BETA VALUES USED AS WEIGHTS

Weight	W1	W2	W3	W4	W5	W6	W7	W8	W9
Alpha	1/8	1/8	1/8	3/16	3/16	3/16	1/16	1/16	1/16
Beta	1/4	3/8	1/8	1/4	3/8	1/8	1/4	3/8	1/8

In Figs. 15 and 16 respectively, the throughput for 75 Mbps background traffic varies between 8-16 Mbps, and the delay varies between 40-100 milliseconds. In Figs. 17-20, due to congestion caused by the background traffic, the highest throughput for 100 and 125 Mbps background traffic was 450 and 375 Kbps respectively, while the lowest throughput was 50 Kbps. Delays for these higher levels of background traffic vary between a few to over 40 milliseconds. The results in general show that while the often used W1/One combination gives the maximum throughput only for 100 Mbps background traffic, it has the minimum delay with both 100 and 125 Mbps background traffic. However, no particular combination of weight and sampling strategy is consistently better than the others. Also, sampling more often does not always give higher throughput or lower delays.

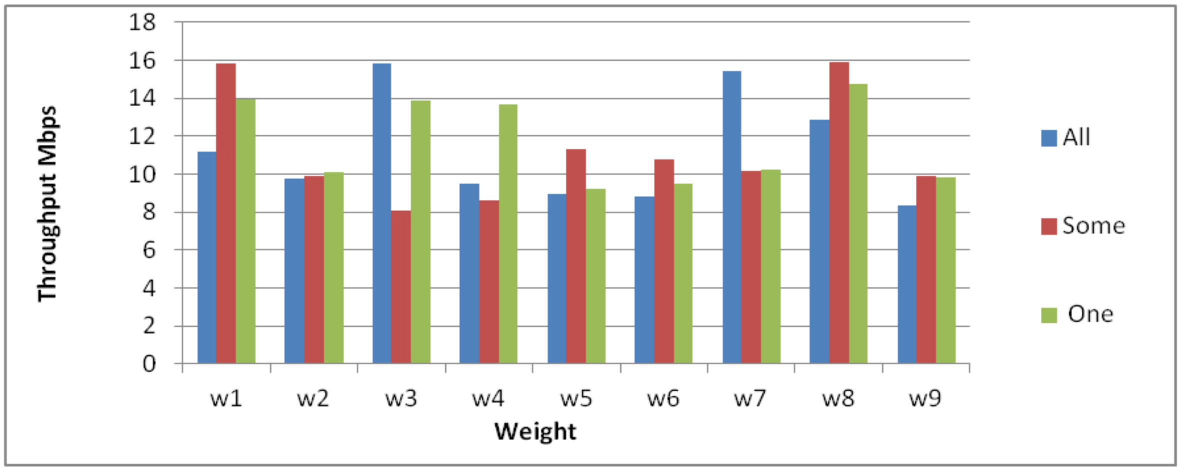

Figure 15. Throughput at 75 Mbps background traffic

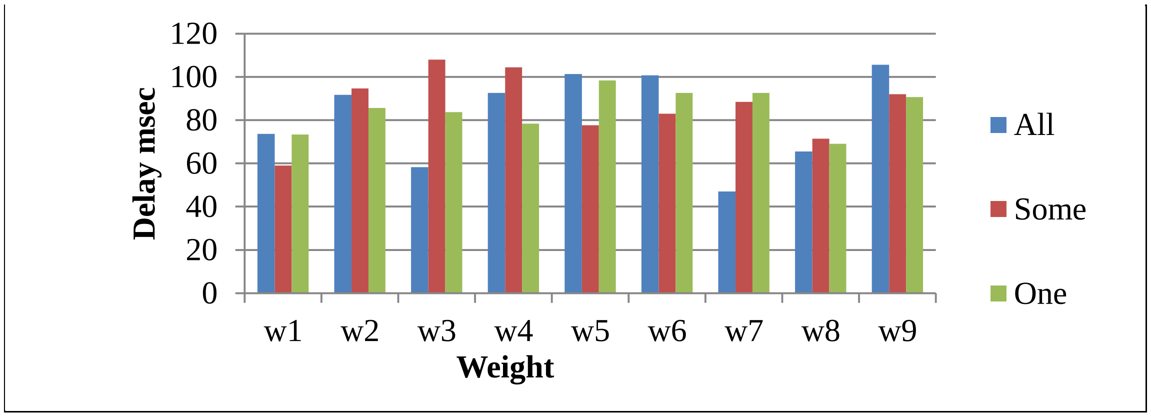

Figure 16. Delay at 75 Mbps background traffic

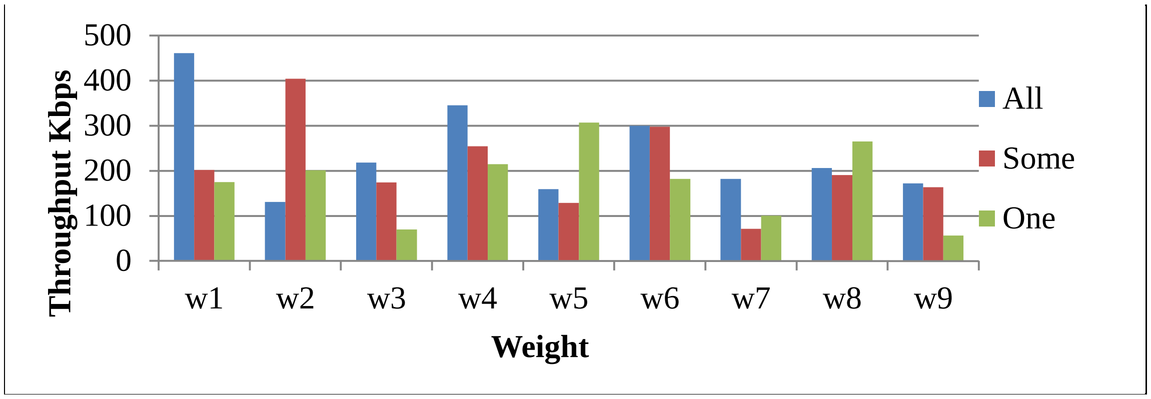

Figure 17. Throughput at 100 Mbps background traffic

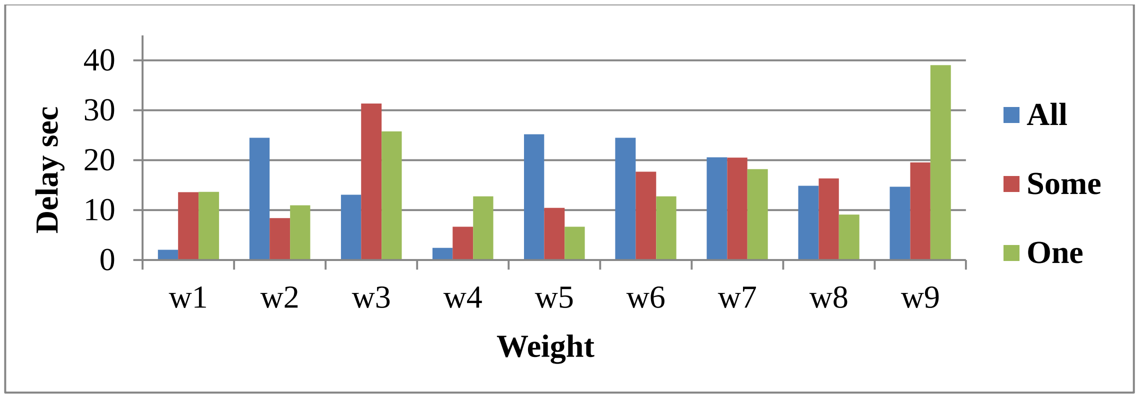

Figure 18. Delay at 100 Mbps background traffic

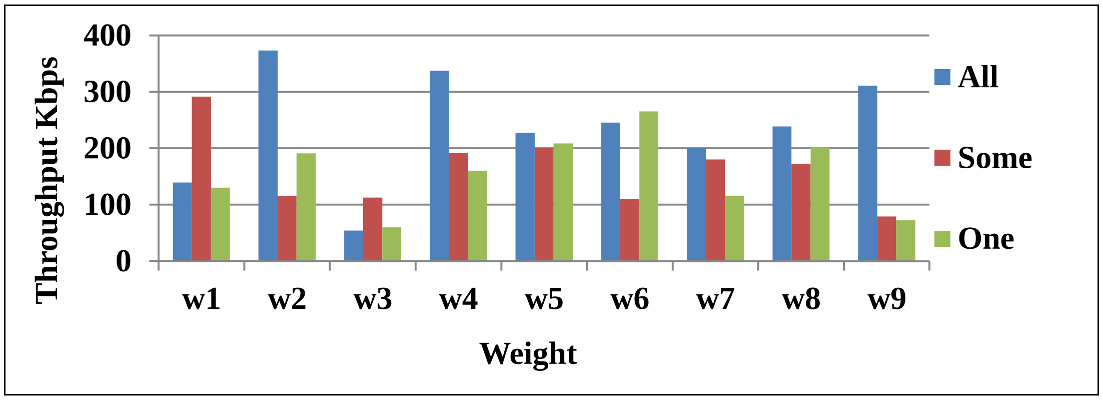

Figure 19. Throughput at 125 Mbps background traffic

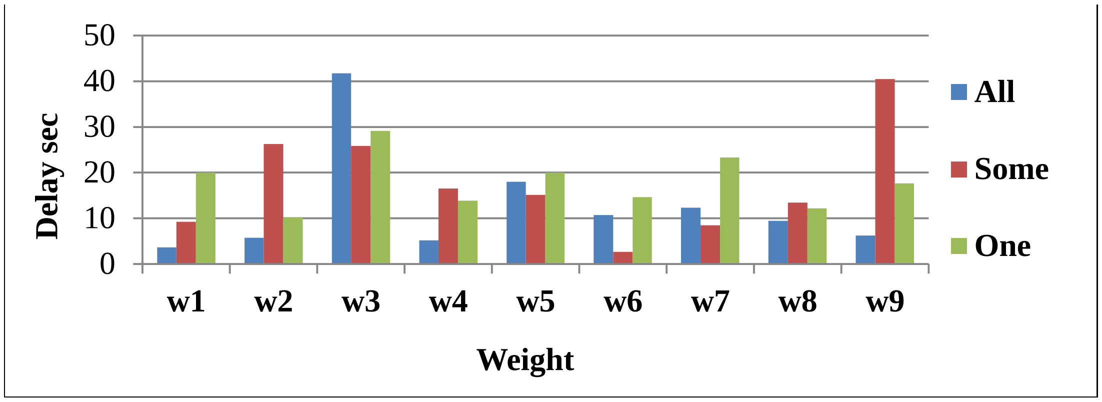

Figure 20. Delay at 125 Mbps background traffic

4.1.3 RTO and RTT

To determine the accuracy of the computed RTO value for the different weights W1-W9, we compare it with the actual value of RTT in each case. As previously, we use the three sampling strategies “some” (S), “one” (O), and “all” (A), and background traffic levels of 75, 100, and 125 Mbps. The results for these sampling strategies and the three weights W1, W5, and W9 are shown in Table II (results for the other combinations were similar). The values in the table are the percentages of packets for which the RTT is within the computed RTO for a given weight/sampling strategy considering the two cases RTO $\leq$ 1 second and RTO $>$ 1 second respectively. The table also shows the percentage of packets whose RTT is less than 1 second and exceeds the computed RTO. Such packets would not be retransmitted if the minimum RTO setting is 1 second as required by [13].

For 75 Mbps background traffic, all packets had RTO $<$ 1 second and the RTT did not exceed the RTO for a majority of the packets (83-100%). For this level of background traffic, the percentage of packets that would need to be retransmitted if the minimum RTO is not set to 1 second is highest (17%) with the W1/One combination. For 100 and 125 Mbps background traffic, although between 1-68% of packets have RTO > 1 second, none of these packets needed to be retransmitted. For these levels of background traffic, for packets that have RTO < 1 second, almost all do not need retransmission even if a minimum RTO setting of 1 second is not used (regardless of the weight/sampling strategy combination). For example, with the W1/One combination, 50% of packets have RTO <1 second and none of the packets need to be retransmitted.

TABLE 6. PERCENTAGE OF PACKETS WITHIN MINIMUM RTO

		RTO ≤ 1 s									RTO > 1 s					
		RTT ≤ RTO			RTT > RTO						RTT < RTO			RTT ≥ RTO		
					RTT < 1 s			RTT ≥ 1 s								
Mbps	**Wt.**	S	O	A	S	O	A	S	O	A	S	O	A	S	O	A
75	**W1**	96	83	96	4	17	4	0	0	0	0	0	0	0	0	0
	W5	93	93	96	7	7	4	0	0	0	0	0	0	0	0	0
	W9	94	100	96	6	0	4	0	0	0	0	0	0	0	0	0
100	**W1**	72	50	98	0	0	1	0	0	0	29	50	1	0	0	0
	W5	86	59	88	0	0	0	0	0	0	14	41	12	0	0	0
	W9	75	32	81	0	0	0	0	0	0	25	68	19	0	0	0
125	**W1**	73	50	62	0	0	0	0	0	0	27	50	38	0	0	0
	W5	80	50	82	0	0	0	0	0	0	20	50	18	0	0	0
	W9	73	50	90	0	0	0	0	4	0	27	46	10	0	0	0

The results of investigating the minimum RTO setting in more detail are shown in Figs. 21 and 22. The graphs show the percentage of packets that would avoid retransmission if the minimum RTO thresholds were set to 0.25, 0.5, 0.75 and 1 second with 100 and 125 Mbps background traffic (the case of 75 Mbps is not shown since the RTO was then always less than 0.25 seconds and the percentage is 100% at all thresholds). For 100 Mbps background traffic (Fig. 21), using the current minimum RTO setting of 1 second, close to 100% of the packets avoid retransmission with the W1/All combination. If a lower minimum RTO setting of 0.5 or 0.75 seconds is used with the W1/All combination, the percentage is essentially the same. However, the percentage drops with the other combinations, and it can be as low as 50% even using a minimum RTO of 1 second with the W1/One combination. At a higher background traffic level of 125 Mbps (Fig. 22), with the W1/All combination, only 60% of packets avoid retransmission with a minimum RTO of 1 second, but there is only a small drop in this percentage using a minimum RTO of 0.75 or 0.5 seconds. With the W1/One combination, the percentage of packets avoiding retransmission is 50% of packets using a minimum RTO of 1 second, but only 35% using a minimum RTO of 0.75 or 0.5 seconds.

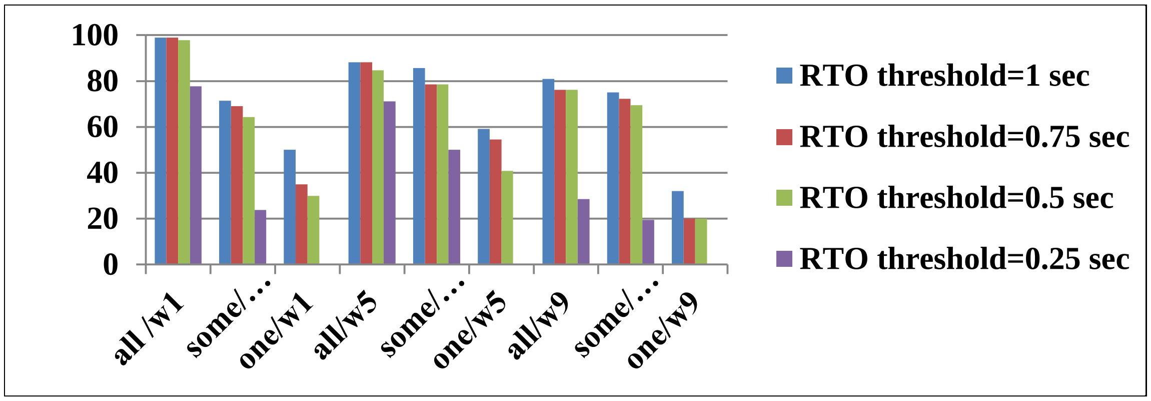

Figure 21. Percentage of packets avoiding retransmission at 100 Mbps Background Traffic

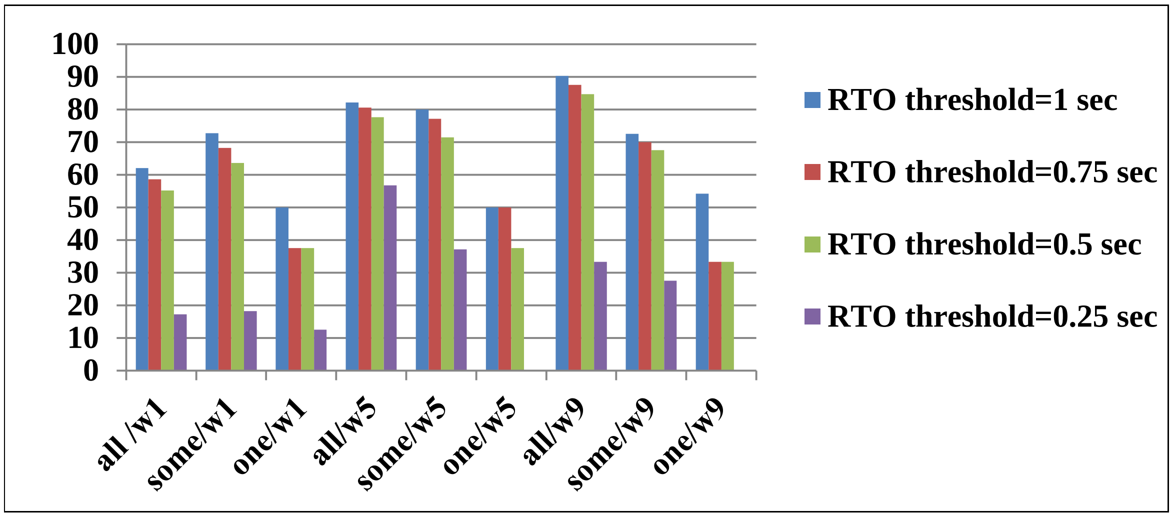

Figure 22. Percentage of packets avoiding retransmission at 125 Mbps Background Traffic

4.1.4 Web server performance

To compare the performance of a bare PC Web server with the standard timer settings in [13] and the Apache and IIS Web servers with their default settings, the goodput (i.e., the application throughput discounting retransmissions) and delay were measured under background traffic of 75, 100, and 125 Mbps using only a pair of the servers at a time i.e., (bare PC, Apache), (Apache, IIS), and (IIS, bare PC). Windows server 2008 (IIS 7) and Apache HTTP Server 2.2.16 running Fedora 12 (Constantine) Linux Kernel 2.6.31.5-127.fc12.i686 were used for the performance comparisons. The bare server used only one sample measurement per RTT (i.e., the sampling strategy "One") in the RTO algorithm. Each experiment was run 6 times and the average values are shown in Figs. 23-28 (there were no significant differences in the values that were averaged).

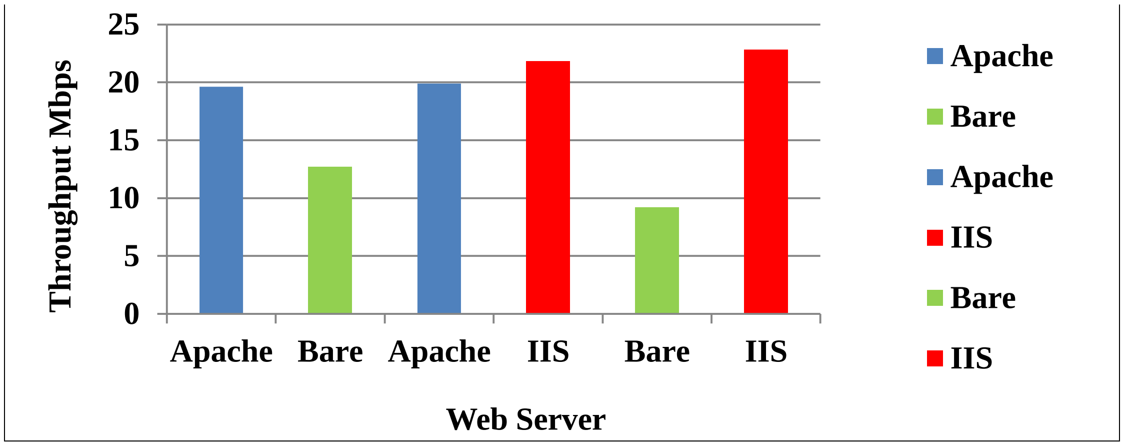

Figure 23. Web server goodput at 75 Mbps background traffic

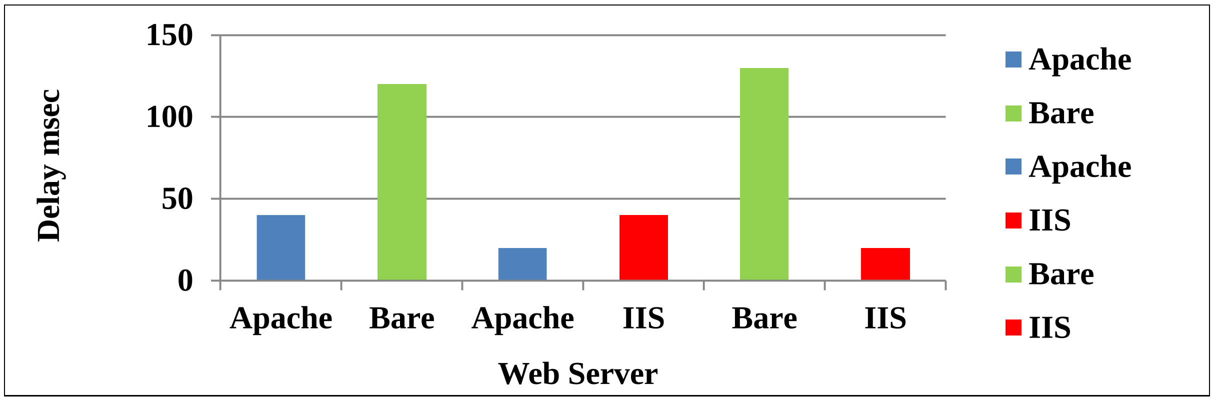

Figure 24. Web server delay at 75 Mbps background traffic

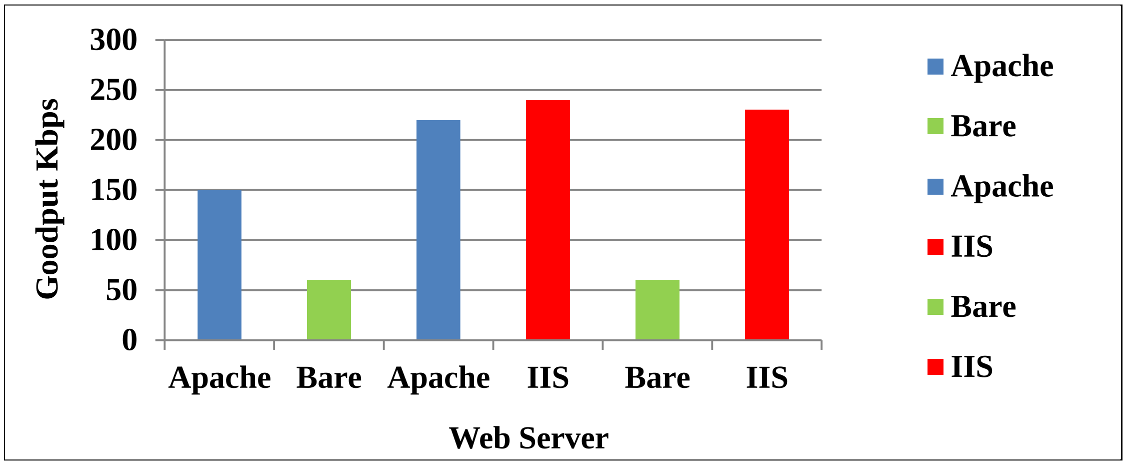

Figure 25. Web server Goodput at 100 Mbps background traffic

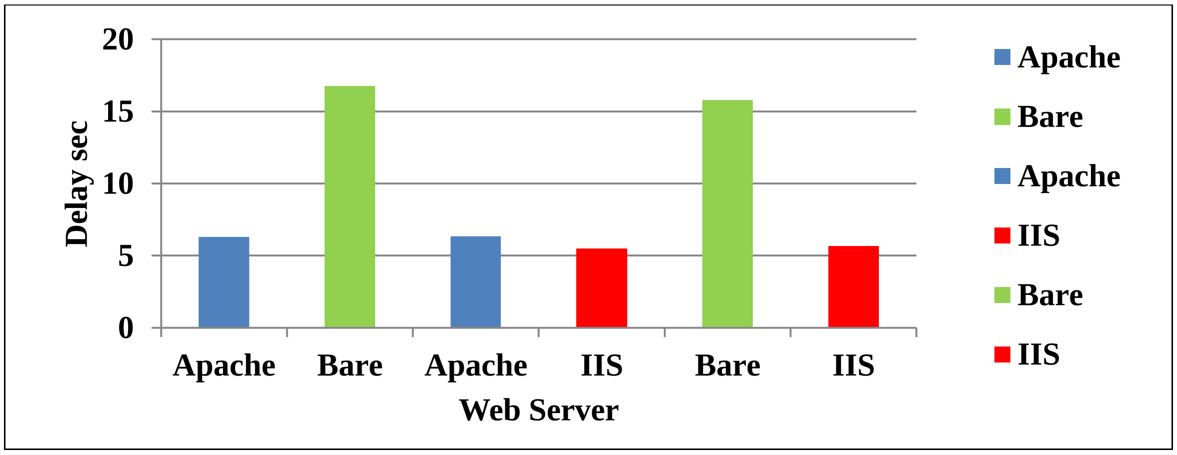

Figure 26. Web server delay at 100 Mbps background traffic

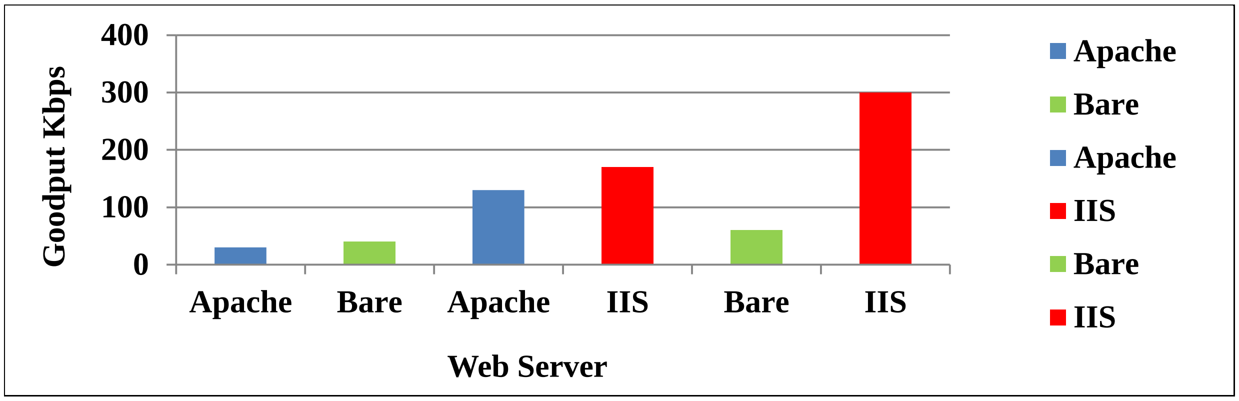

Figure 27. Web server Goodput at 125 Mbps background traffic

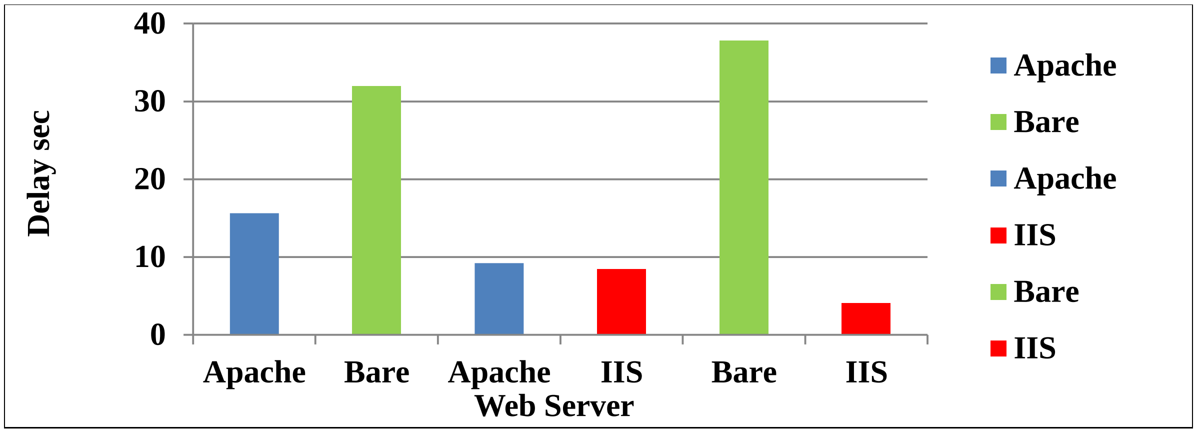

Figure 28. Web server delay at /125 Mbps background traffic

CHAPTER 5

PERFORMANCE OF AN IPV6 WEB SERVER UNDER CONGESTION

5.1 Network Architecture and experiments

The network for our experiments is shown in Fig. 1. The test LAN consists of five Ethernets connected by routers. All the routers run Fedora 12 Kernel Linux 2.6.31.5-127.fc12.i686, and the network interface cards are 1 Gbps except for a 100 Mbps card on the client side of router R1. The Ethernet switches used are 1 Gbps except for switch S0, which is 100 Mbps. The 100 Mbps link and network act as a bottleneck to create congestion.

Two pairs of machines running MGEN [15] generate the background TCP and UDP congestion traffic. One machine (Dell OPTEPLEX GX260, CentOS Version 2.16.0) is a TCP source and UDP sink, and its peer (Dell OPTEPLEX GX260, Windows XP Professional 2002 SP3) is a TCP sink and UDP source to generate background IPv4 traffic. The other pair (same specifications as the first pair) serves to generate background IPv6 traffic in a similar manner. An Apache HTTP Server 2.2.16 running Fedora 12 (Constantine) Kernel Linux 2.6.31.5-127.fc12.i686 or a bare PC Web server is used as the Web server. A Firefox browser version 3.5.4 running Fedora Linux kernel 2.6.31.5-127.fc12.i686 (on the machine labeled as client in Fig. 1) makes individual requests to the Web server. The clients, servers, and routers run on Dell OPTEPLEX GX 520 PCs.

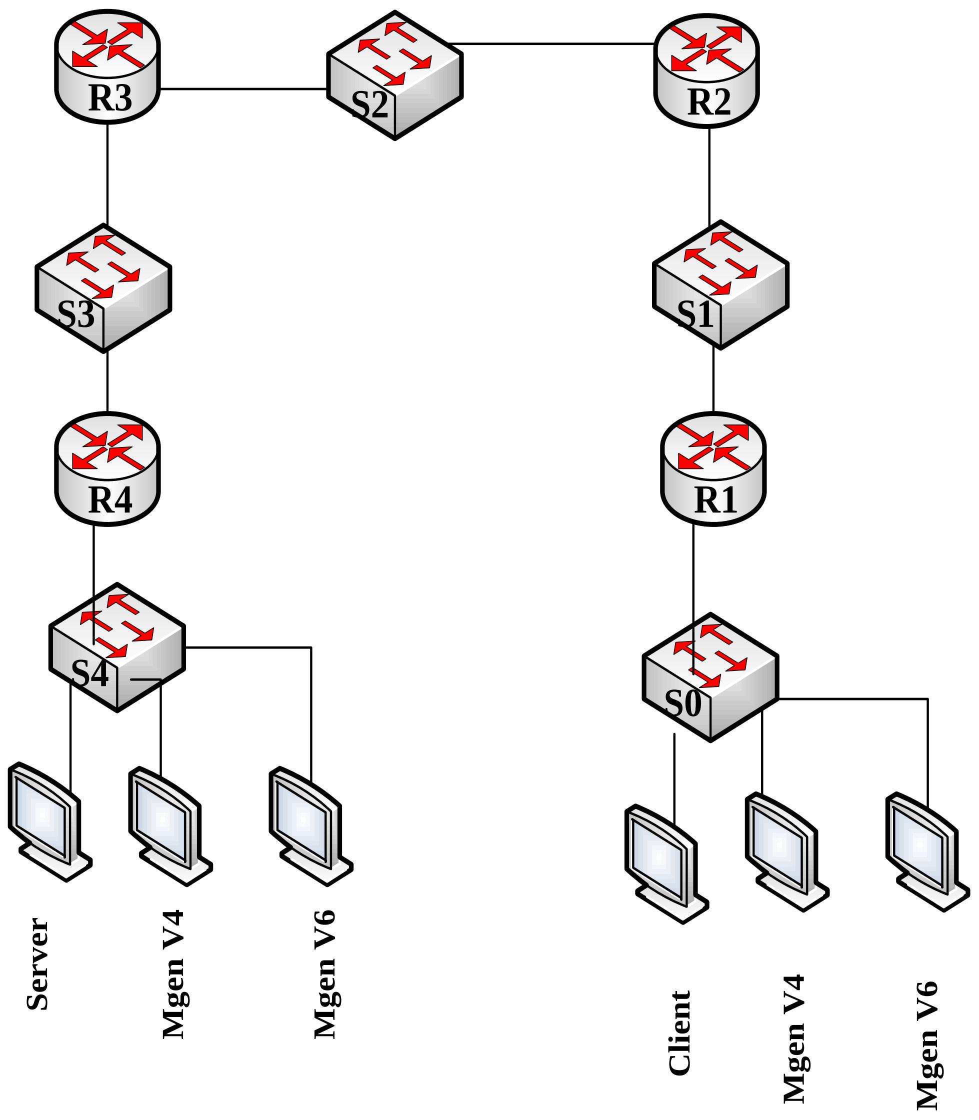

Figure 29. Test network2

The background traffic consists of a mix of 10% UDP and 90% TCP traffic, which reasonably represents the traffic composition for these protocols in the current Internet. Two different levels of traffic are used: 75 Mbps (moderate congestion) and 100 Mbps (higher congestion). Each of these levels can be generated in three ways using different percentages of TCP/UDP traffic over IPv4 and IPv6. They are labeled as congestion levels C1B, C2B, and C3B, and C4B, C5B, and C6B respectively. Table I shows the specific TCP/UDP percentages for each level, where 4 or 6 refers to IPv4 or IPv6 respectively. The amounts of TCP data carried in the MGEN packets are 1440 and 1460 bytes respectively for IPv6 and IPv4, and it is configured to use 10 flows at different rates to achieve the desired levels of congestion.

The throughput and delay when the Firefox Web browser requests the 320 KB file from the Apache or bare PC Web server were determined by using the network protocol analyzer Wireshark (with port mirroring) at the server side i.e., connected to switch S4. The throughput ignores retransmissions, but considers both incoming and outgoing traffic associated with a single request. From the Wireshark traces, the values of the delay were computed by using the time stamps for the HTTP Get request and the last valid ack sent by the client (before the FIN+ACK). While the throughput and delay are related, the delay is the delay for the data transfer only i.e., it is not the total delay for the request since it is not measured from the TCP SYN. Each experiment was repeated ten times (for each of IPv4 and IPv6), and the average values of the delay were used.

TABLE 7. BACK GROUND TRAFFIC CONDITIONS

Condition	TCP4 %	TCP6 %	UDP4 %	UDP6 %	Total Mbps
C1B	45	45	5	5	75
C2B	22.5	67.5	2.5	7.5	75
C3B	67.5	22.5	7.5	2.5	75
C4B	45	45	5	5	100
C5B	22.5	67.5	2.5	7.5	100
C6B	67.5	22.5	7.5	2.5	100

5.2 Results

5.2.1 Apache Throughput

The throughput associated with a single browser request to the Apache server was obtained as described above. The results are shown in Fig. 2. It is seen that the throughput for IPv4 is slightly higher than for IPv6 except for congestion level C3B when they are approximately equal (IPv6 throughput is higher, but the difference is only about 0.4 Mbps). The other differences range from approximately 1 Mbps (for congestion level C6B) to 7 Mbps (for congestion level C2B).

It can also be seen from the figure that throughput values range from about 20-27 Mbps for IPv6 and from about 22-31 Mbps for IPv4. The highest throughput is with congestion level C3B and C2B for IPv6 and IPv4 respectively, and the lowest throughput is with level C4B for both IPv6 and IPv4. So the IPv6 throughput is highest when the congestion traffic is lower (75 Mbps) and its percentage of IPv6 traffic is lower (25%), and it is lowest when the congestion traffic is higher (100 Mbps) and its percentage of IPv6 traffic is equitable (50%). Similarly, the IPv4 throughput is highest when the congestion traffic is lower and its percentage of IPv4 traffic is lower, and it is lowest when the congestion traffic is higher and its percentage of IPv4 traffic is equitable. For both IPv6 and IPv4, the throughput is highest when congestion traffic is lower and like traffic is lower, and it is lowest when congestion traffic is higher and like traffic is 50% or more. Also, the low percentage of UDP traffic appears to have had a negligible impact on throughput as would be expected.

5.2.2 Apache Delay

In the absence of congestion traffic, the delay for the file transfer from Apache to the browser was found to be 48 milliseconds for both IPv4 and IPv6. Fig. 3 shows the transfer delay when the browser makes the request under each of the six congestion levels C1B-C6B. The delays during congestion are much larger, between 84-103 ms for IPv4 and between 105-121 ms for IPv6. Also, IPv6 delays are always larger, and the delay difference between the IPv4 and IPv6 delays (for a given congestion level) varies from 6-32 ms. The difference between IPv6 and IPv4 delays are largest for congestion levels C2B and C5B, which have larger percentages of IPv6 traffic (75%), and smallest for level C3B, which has lower congestion traffic (75 Mbps) and a lower percentage of IPv6 traffic (25%).

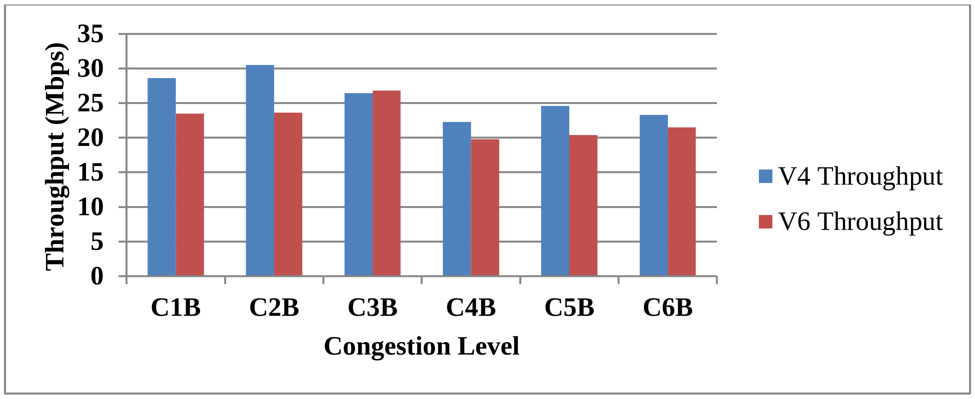

Figure 30. Apache throughput under congestion

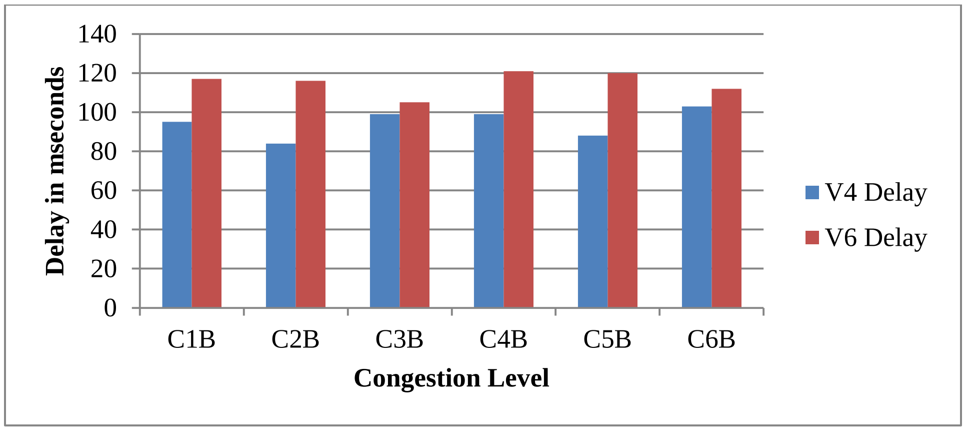

Figure 31. Apache delay under congestion

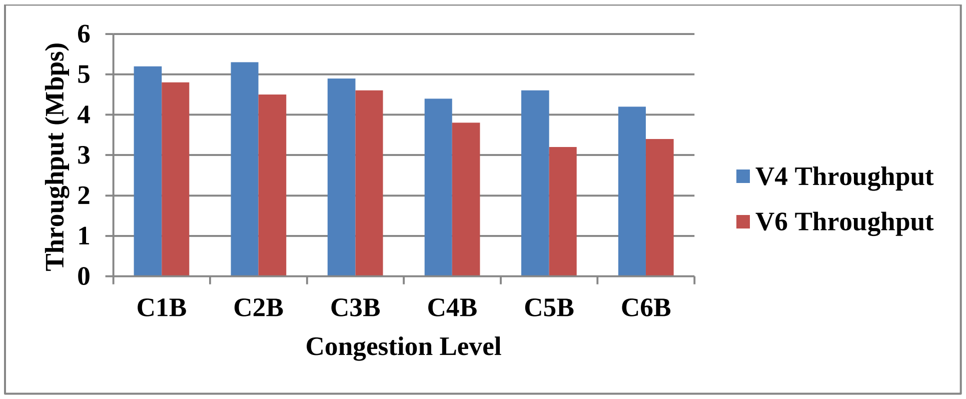

Figure 32. Bare server throughput under congestion

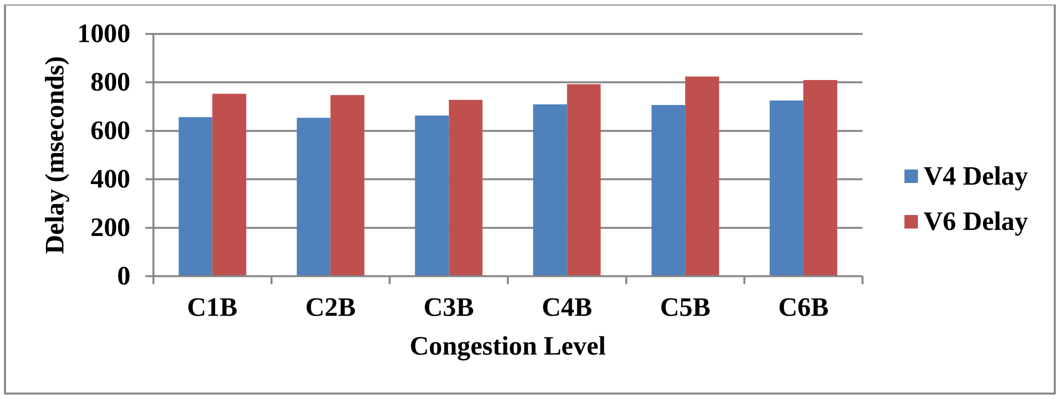

Figure 33. Bare server delay under congestion

The highest delay is with congestion level C4B and C6B for IPv6 and IPv4 respectively, and the lowest delay is with level C3B and C2B for IPv6 and IPv4 respectively. It can be seen that the IPv6 delay is highest when the congestion traffic is higher (100 Mbps) and its percentage of IPv6 traffic is equitable (50%), and it is lowest when the congestion traffic is lower (75 Mbps) and its percentage of IPv6 traffic is lower (25%). Similarly, the IPv4 delay is highest when the congestion traffic is higher and its percentage of IPv4 traffic is higher, and it is lowest when the congestion traffic is lower and its percentage of IPv4 traffic is lower. For both IPv6 and IPv4, the delay is highest when congestion traffic is higher and like traffic is 50% or more, and it is lowest when congestion traffic is lower and like traffic is lower.

The above results using the Apache server and an ordinary browser suggest that the throughput and delay for a single request over IPv6 or IPv4 in a network with mixed congestion traffic are affected by two factors. First, the rate of congestion traffic, and second, by the percentage of like traffic in the congestion mix. It should also be noted that these results are for the case when the server only receives a single request from a browser i.e., there are no additional delays at the server due to any other requests.

5.2.3 Bare PC Web Server Throughput

We now consider bare PC Web server performance. Fig. 4 shows the throughput for a browser request from Firefox under the six congestion levels. It can be seen that throughput is much lower than for the Apache server under the same congestion levels. The throughput for IPv4 is slightly higher than for IPv6 as seen above for the Apache server. It ranges from 3.4-4.8 Mbps for IPv6 and from 4.2-5.3 Mbps for IPv4. The differences between IPv6 throughputs for different congestion levels are small. For example, the throughput difference for levels C2B and C3B with IPv6, and for levels C1B and C2B with IPv4 is only 0.1 Mbps. The throughput differences range from 0.3 Mbps (for congestion level C3B) to 1.4 Mbps (for congestion level C5B). It can be seen that the differences between IPv4 and IPv6 throughput under a given level of congestion are much smaller than for Apache. The IPv6 throughput is highest for congestion level C1B and lowest for level C5B, while the IPv4 throughput is highest for congestion level C2B and lowest for level C6B.

The likely reason that the bare PC Web server has lower throughput than Apache under congestion is the absence of TCP optimizations. In particular, the bare PC server does not use selective acks (i.e. TCP SACK) or fast retransmit/recovery, and retransmits all the data after waiting for a timeout in the event of a loss. However, the bare PC Web server implementation does not have separate IPv4 and IPv6 stacks unlike a conventional server such as Linux. Instead, it uses a single RCV (Receive) task (in a single thread of execution) to process an incoming IP packet regardless of whether it is an IPv6 packet or an IPv4 packet. Also, the HTTP and TCP code in the server is intertwined. This implies that there is little difference in overhead when processing the two types of IP packets. However, more studies with a bare PC server that implements TCP optimizations is needed to determine the extent of possible throughput improvement for IPv6 and IPv4 due to eliminating operating system overhead. Similarly, the impact on bare PC server performance due to not implementing any congestion control mechanisms is not known.

The IPv6 throughput is highest when the congestion traffic is lower (75 Mbps) and its percentage of IPv6 traffic is equitable (50%), and it is lowest when the congestion traffic is higher (100 Mbps) and its percentage of IPv6 traffic is higher (75%). Similarly, the IPv4 throughput is highest when the congestion traffic is lower and its percentage of IPv4 traffic is lower, and it is lowest when the congestion traffic is higher and its percentage of IPv4 traffic is higher. For both IPv6 and IPv4, the throughput is highest when congestion traffic is lower and like traffic is 50% or less, and it is lowest when congestion traffic is higher and like traffic is higher. This is similar to the results seen for Apache.

5.2.4 Bare Web Server Delay

The delays to transfer the 320 KB file to the browser from the bare PC server in the absence of congestion traffic are 99 milliseconds and 116 milliseconds over IPv4 and IPv6 respectively (more than double the delays for Apache). When there is congestion traffic, delays range from 727-824 ms for IPv6 and from 654-725 ms for IPv4, which are much larger than the corresponding delays for Apache. Also, it can be seen that IPv6 delay are larger than the IPv4 delays as for Apache, but the delay differences between IPv6 and IPv4 for all congestion levels are now much higher than for Apache. These differences range from 64-118 ms. As with lowered throughput, the increased delays are likely due to the absence of any TCP optimizations.

The difference between IPv6 and IPv4 delays are largest for congestion levels C5B, which has a larger percentage of IPv6 traffic (75%), and smallest for level C3B, which has lower congestion traffic (75 Mbps) and a lower percentage of IPv6 traffic (25%). The lowest delay is with congestion level C3B for IPv6 and with level C2B for IPv4, and the highest delay is with congestion level C5B for IPv6 and level C6B for IPv4. It can be seen that the IPv6 delay is highest when the congestion traffic is higher (100 Mbps) and its percentage of IPv6 traffic is higher (75%), and it is lowest when the congestion traffic is lower (75 Mbps) and its percentage of IPv6 traffic is lower (25%). Similarly, the IPv4 delay is highest when the congestion traffic is higher and its percentage of IPv4 traffic is higher, and it is lowest when the congestion traffic is lower and its percentage of IPv4 traffic is lower. For both IPv6 and IPv4, the delay is highest when congestion traffic is higher and like traffic is higher, and it is lowest when congestion traffic is lower and like traffic is lower.

These results for bare PC server throughput and delay are similar to the results for the Apache server, although the throughput is much lower and the delays are much higher. As with Apache, we see that the throughput and delay depend on both the congestion rate of congestion traffic and the percentage of like traffic causing the congestion.

CHAPTER 6

CONCLUSION

We studied the performance of conventional and bare PC Web servers under different levels of congestion. We first conducted experiments in a test network to study the impact of two widely implemented TCP congestion control variants Compound TCP and Cubic TCP on Web server performance using the popular IIS and Apache Web servers. Congestion was created by generating a mix of background traffic and different workloads, and performance was measured using server delay for a single browser request, throughput measured at the server and client ends, and shared bandwidth percentage. The Apache server with Cubic TCP has less delay than the IIS server with Compound TCP for large and medium files, but not for small files. Throughput for IIS is slightly higher than throughput for Apache regardless of the congestion scenarios that were used in the experiments. For experiments with like servers, IIS shares bandwidth more equitably than Apache and for experiments with unlike servers, the latter adopts a conservative approach that limits its performance. When IIS competes with Apache for bandwidth under congestion, although both get a roughly equal share of the allowed bandwidth, the delay for Apache rises to an unacceptably high level.

We next studied the impact of changing the values of alpha and beta in the TCP retransmission algorithm using recently recommended initial timer settings. Our studies used a bare PC Web server with no congestion control mechanisms and no TCP optimizations. We also studied the performance of the Web server for various values of the minimum retransmission timeout, and compared the performance of the bare server with the Apache and IIS Web servers.

Our results show that no values of alpha and beta are better than others, and standard timer settings with no SACK and no congestion control degrade performance for some traffic levels.

We finally studied the performance of IPv6 Web servers under different levels of congestion. Studies were conducted in a test LAN with several routers, and used a conventional Apache Web server and a bare PC Web server with no operating system. HTTP requests over IPv6 and IPv4 were sent to the servers using an ordinary Web browser. The results for both servers show that throughput over IPv6 throughput is slightly lower than or approximately equal to IPv4 throughput depending on the congestion level, whereas delays are much higher over IPv6 than over IPv4. For all congestion levels, bare PC server throughput and delay are significantly worse than for Apache due to not implementing any TCP optimizations. The results of this study show that the performance of an IPv6 Web server for requests over IPv6 and IPv4 depend on both the congestion traffic rate and the percentage of like IP traffic.

APPENDIX

Chapter 3 data tables

File size Kbytes	TCP Stress Mbps	TCP BT Mbps	Server Captured Mbps	Delay Seconds	NRT Mbps
504	**42.20**	**37.64**	**55.34**	**16.27**	**50.94**
320	**42.15**	**37.58**	**55.24**	**16.41**	**50.58**
160	**42.25**	**37.53**	**54.64**	**7.34**	**50.62**

TABLE 8. PERFORMANCE UNDER C1 CONDITION (APACHE-APACHE SCENARIO)

File size Kbytes	TCP Stress Mbps	TCP BT Mbps	Server Captured Mbps	Delay Seconds	NRT Mbps
504	**24.00**	**52.34**	**32.48**	**29.09**	**29.39**
320	**23.98**	**52.31**	**32.37**	**9.60**	**29.10**
160	**23.81**	**52.10**	**32.18**	**9.49**	**29.07**

TABLE 9. PERFORMANCE UNDER C2 CONDITION (APACHE-APACHE SCENARIO)

File size Kbytes	TCP Stress Mbps	TCP BT Mbps	Server Captured Mbps	Delay Seconds	NRT Mbps
504	**51.37**	**28.80**	**65.83**	**21.88**	**59.82**
320	**51.52**	**28.83**	**65.38**	**16.83**	**59.67**
160	**51.45**	**28.72**	**66.63**	**9.89**	**60.41**

TABLE 10. PERFORMANCE UNDER C3 CONDITION (APACHE-APACHE SCENARIO)

File size Kbytes	TCP Stress Mbps	TCP BT Mbps	Server Captured Mbps	Delay Seconds	NRT Mbps
504	**42.18**	**34.83**	**63.92**	**31.01**	**52.62**
320	**42.12**	**34.86**	**65.21**	**26.19**	**53.99**
160	**41.88**	**34.77**	**66.59**	**5.53**	**56.16**

TABLE 11. PERFORMANCE UNDER C1 CONDITION (IIS-IIS SCENARIO)

File size Kbytes	TCP Stress Mbps	TCP BT Mbps	Server Captured Mbps	Delay Seconds	NRT Mbps
504	**22.17**	**53.90**	**33.52**	**30.08**	**28.29**
320	**21.97**	**53.92**	**32.53**	**19.73**	**27.53**
160	**22.22**	**54.02**	**33.27**	**8.35**	**28.35**

TABLE 12. PERFORMANCE UNDER C2 CONDITION (IIS-IIS SCENARIO)

File size Kbytes	TCP Stress Mbps	TCP BT Mbps	Server Captured Mbps	Delay Seconds	NRT Mbps
504	**50.44**	**26.65**	**79.96**	**34.35**	**65.33**
320	**49.39**	**26.46**	**77.23**	**26.01**	**63.77**
160	**50.73**	**26.35**	**76.87**	**7.52**	**64.17**

TABLE 13. PERFORMANCE UNDER C3 CONDITION (IIS-IIS SCENARIO)

Condition	TCP Stress Mbps	TCP BT Mbps	Server Captured Mbps	Delay Seconds	NRT Mbps
C1	**45.34**	**33.72**	**64.25**	**28.11**	**54.96**
C2	**24.28**	**51.86**	**33.57**	**16.70**	**30.52**
C3	**51.43**	**25.76**	**78.23**	**21.71**	**65.01**

TABLE 14. PERFORMANCE UNDER DIFFERENT CONDITIONS FOR FILE 504KBYTES (IIS-APACHE SCENARIO)

Condition	TCP Stress Mbps	TCP BT Mbps	Server Captured Mbps	Delay Seconds	NRT Mbps
C1	**41.77**	**38.73**	**56.28**	**24.68**	**49.24**
C2	**21.84**	**53.99**	**32.83**	**42.42**	**27.33**
C3	**51.14**	**29.40**	**64.43**	**21.34**	**58.99**

TABLE 15. PERFORMANCE UNDER DIFFERENT CONDITIONS FOR FILE 504KBYTES (APACHE-IIS SCENARIO)

S3 Mbps	S4 Mbps	S1 Mbps	Delay S1 seconds	S2 Mbps	Delay S2 seconds
36.95	**36.74**	**0.10**	**32.24**	**0.09**	**36.00**

TABLE 16. SERVERS PERFORMANCE (S1=S2= APACHE; S3=S4=IIS)

S3 Mbps	S4 Mbps	S1 Mbps	Delay S1 seconds	S2 Mbps	Delay S2 seconds
35.81	**36.06**	**0.41**	**8.89**	**0.40**	**7.40**

TABLE 17. SERVERS PERFORMANCE (S1=S2= IIS; S3=S4=APACHE)

S3 Mbps	S4 Mbps	S1 Mbps	Delay S1 seconds	S2 Mbps	Delay S2 seconds
36.02	**37.49**	**0.19**	**15.62**	**0.44**	**7.78**

TABLE 18. SERVERS PERFORMANCE (S2=S4= IIS; S1=S3=APACHE)

S1Mbps	S1 Delay seconds	S2 Mbps	S2 Delay seconds	S3 Mbps	S1 NRT Mbps	S2 NRT Mbps
38.55	**34.48**	**38.43**	**41.03**	**32.39**	**32.65**	**32.68**

TABLE 19. SEVERS PERFORMANCE (S1=S2= APACHE; S3=IIS)

C1 stress Mbps	C2 stress Mbps	Cbt Mbps
26.18	**26.42**	**25.35**

TABLE 20. CLIENTS PERFORMANCE (S1=S2= APACHE; S3=IIS)

S1Mbps	S1 Delay seconds	S2 Mbps	S2 Delay seconds	S3 Mbps	S1 NRT Mbps	S2 NRT Mbps
38.42	**22.37**	**38.60**	**23.05**	**34.69**	**32.55**	**32.90**

TABLE 21. SEVERS PERFORMANCE (S1=S2=IIS; S3=APACHE)

C1 stress Mbps	C2 stress Mbps	Cbt Mbps
26.04	**26.45**	**25.69**

TABLE 22. CLIENTS PERFORMANCE (S1=S2=IIS; S3=APACHE)

S1 Mbps	S1 Delay seconds	S2 Mbps	S2 Delay seconds	S3 Mbps	S1 NRT Mbps	S2 NRT Mbps
40.08	**19.69**	**38.84**	**21.45**	**35.42**	**33.42**	**32.13**

TABLE 23. SEVERS PERFORMANCE (S1=IIS; S2=APACHE; S3=APACHE)

C1 stress Mbps	C2 stress Mbps	Cbt Mbps
26.40	**26.08**	**25.71**

TABLE 24. CLIENTS PERFORMANCE (S1=IIS; S2=APACHE; S3=APACHE)

Chapter 4 data tables

Throughput Mbps	Delay Milliseconds	Weight
15.85	**59**	W1
9.92	**94.67**	W2
8.08	**108**	W3
8.62	**104.5**	W4
11.29	**77.67**	W5
10.77	**83**	W6
10.19	**88.5**	W7
15.88	**71.5**	W8
9.87	**92**	W9

TABLE 25. THROUGHPUT AND DELAY AT 75MBPS FOR “SOME” SCENARIO

Throughput Mbps	Delay Milliseconds	Weight
0.20	**13578**	W1
0.40	**8403**	W2
0.17	**31341**	W3
0.25	**6684**	W4
0.13	**10452.5**	W5
0.30	**17688.5**	W6
0.07	**20526.5**	W7
0.19	**16356.5**	W8
0.16	**19564**	W9

TABLE 26. THROUGHPUT AND DELAY AT 100MBPS FOR “SOME” SCENARIO

Throughput Mbps	Delay Milliseconds	Weight
0.29	**9261**	W1
0.12	**26295**	W2
0.11	**25857.5**	W3
0.19	**16563.5**	W4
0.20	**15165.5**	W5
0.11	**2653.046**	W6
0.18	**8510.5**	W7
0.17	**13452**	W8
0.16	**19564**	W9

TABLE 27. THROUGHPUT AND DELAY AT 125MBPS FOR "SOME" SCENARIO

Throughput Mbps	Delay Milliseconds	Weight
13.92	**73.33**	W1
10.07	**85.67**	W2
13.88	**83.67**	W3
13.70	**78.33**	W4
9.23	**98.33**	W5
9.48	**92.67**	W6
10.25	**92.67**	W7
14.77	**69.00**	W8
9.82	**90.67**	W9

TABLE 28. THROUGHPUT AND DELAY AT 75MBPS FOR "ONE" SCENARIO

Throughput Mbps	Delay Milliseconds	Weight
0.18	**13663.33**	W1
0.20	**10993.00**	W2
0.07	**25763.00**	W3
0.21	**12754.67**	W4
0.31	**6679.00**	W5
0.18	**12777.67**	W6
0.10	**18191.33**	W7
0.27	**9079.67**	W8
0.06	**39051.67**	W9

TABLE 29. THROUGHPUT AND DELAY AT 100MBPS FOR "ONE" SCENARIO

Throughput Mbps	Delay Milliseconds	Weight
0.13	**20048.00**	W1
0.19	**10219.00**	W2
0.06	**29137.33**	W3
0.16	**13885.50**	W4
0.21	**20024.50**	W5
0.27	**14646.00**	W6
0.12	**23354.00**	W7
0.20	**12206.67**	W8
0.07	**17654.00**	W9

TABLE 30. THROUGHPUT AND DELAY AT 125MBPS FOR "ONE" SCENARIO

Throughput Mbps	Delay Milliseconds	Weight
11.16	**73.67**	W1
9.75	**91.67**	W2
15.82	**58.33**	W3
9.50	**92.67**	W4
8.92	**101.33**	W5
8.82	**100.67**	W6
15.43	**47.00**	W7
12.87	**65.50**	W8
8.34	**105.67**	W9

TABLE 31. THROUGHPUT AND DELAY AT 75MBPS FOR “ALL” SCENARIO

Throughput Mbps	Delay Milliseconds	Weight
0.461	**2073.67**	W1
0.13	**24501.67**	W2
0.22	**13076.33**	W3
0.35	**2453.06**	W4
0.16	**25166.67**	W5
0.30	**24501.67**	W6
0.18	**20555.00**	W7
0.21	**14841.00**	W8
0.17	**14703.00**	W9

TABLE 32. THROUGHPUT AND DELAY AT 100MBPS FOR “ALL” SCENARIO

Throughput Mbps	Delay Milliseconds	Weight
0.14	**3679.67**	W1
0.37	**5780.00**	W2
0.05	**41707.67**	W3
0.34	**5170.50**	W4
0.23	**18016.00**	W5
0.25	**10722.33**	W6
0.20	**12337.33**	W7
0.24	**9448.50**	W8
0.31	**6253.50**	W9

TABLE 33. THROUGHPUT AND DELAY AT 125MBPS FOR "ALL" SCENARIO

Threshold	All /w1	Some /w1	One /w1	All /w5	Some /w5	One /w5	All /w9	Some /w9	One /w9
1.00 second	**98.89**	**71.43**	**50.00**	**88.14**	**85.71**	**59.09**	**80.95**	**75.00**	**32.00**
0.75 second	**98.89**	**69.05**	**35.00**	**88.14**	**78.57**	**54.55**	**76.19**	**72.22**	**20.00**
0.50 second	**97.78**	**64.29**	**30.00**	**84.75**	**78.57**	**40.91**	**76.19**	**69.44**	**20.00**
0.25 second	**77.78**	**23.81**	**0.00**	**71.19**	**50.00**	**0.00**	**28.57**	**19.44**	**0.00**

TABLE 34. PERCENTAGE OF PACKETS AVOIDING RETRANSMISSION AT 100MBPS FOR DIFFERENT RTO THRESHOLDS

Threshold	All /w1	Some /w1	One /w1	All /w5	Some /w5	One /w5	All /w9	Some /w9	One /w9
1.00 second	**62.07**	**72.73**	**50.00**	**82.09**	**80.00**	**50.00**	**90.28**	**72.50**	**54.17**
0.75 second	**58.62**	**68.18**	**37.50**	**80.60**	**77.14**	**50.00**	**87.50**	**70.00**	**33.33**
0.50 second	**55.17**	**63.64**	**37.50**	**77.61**	**71.43**	**37.50**	**84.72**	**67.50**	**33.33**
0.25 second	**17.24**	**18.18**	**12.50**	**56.72**	**37.14**	**0.00**	**33.33**	**27.50**	**0.00**

TABLE 35. PERCENTAGE OF PACKETS AVOIDING RETRANSMISSION AT 125MBPS FOR DIFFERENT RTO THRESHOLDS

Background traffic	75Mbps	100Mbps	125Mbps
Apache Goodput Mbps	**19.63**	**0.15**	**0.03**
Bare Goodput Mbps	**12.68**	**0.06**	**0.04**
Apache Delay seconds	**0.04**	**6.30**	**15.61**
Bare Delay seconds	**0.12**	**16.76**	**31.99**

TABLE 36. APACHE SERVER AND BARE PC SIDE BY SIDE AT DIFFERENT LEVELS OF BACKGROUND TRAFFIC

Background traffic	75Mbps	100Mbps	125Mbps
Apache Goodput Mbps	**19.90**	**0.22**	**0.13**
IIS Goodput Mbps	**21.81**	**0.24**	**0.17**
Apache Delay seconds	**0.02**	**6.33**	**9.23**
IIS Delay seconds	**0.04**	**5.50**	**8.42**

TABLE 37. APACHE AND IIS SERVERS SIDE BY SIDE AT DIFFERENT LEVELS OF BACKGROUND TRAFFIC

Background traffic	75Mbps	100Mbps	125Mbps
Bare Goodput Mbps	**9.19**	**0.06**	**0.06**
IIS Goodput Mbps	**22.82**	**0.23**	**0.30**
Bare Delay seconds	**0.13**	**15.76**	**37.84**
IIS Delay seconds	**0.02**	**5.67**	**4.13**

TABLE 38. IIS SERVER AND BARE PC SIDE BY SIDE AT DIFFERENT LEVELS OF BACKGROUND TRAFFIC

Chapter5 data tables

Congestion Level	**V4 Throughput**	**V6 Throughput**
C1B	**5.2**	**4.8**
C2B	**5.3**	**4.5**
C3B	**4.9**	**4.6**
C4B	**4.4**	**3.8**
C5B	**4.6**	**3.2**
C6B	**4.2**	**3.4**

TABLE 39. BARE THROUGHPUT IN MBPS

Congestion Level	**V4 Delay**	**V6 Delay**
C1B	**656**	**752**
C2B	**654**	**748**
C3B	**663**	**727**
C4B	**709**	**792**
C5B	**706**	**824**
C6B	**725**	**809**

TABLE 40. BARE DELAY IN MS

Congestion Level	**V4 Throughput**	**V6 Throughput**
C1B	28.6	23.5
C2B	30.5	23.6
C3B	26.44	26.8
C4B	22.28	19.8
C5B	24.6	20.4
C6B	23.28	21.5

TABLE 41. APACHE THOURGHPUT IN MBPS

Congestion Level	**V4 Delay**	**V6 Delay**
C1B	0.095	0.117
C2B	0.084	0.116
C3B	0.099	0.105
C4B	0.099	0.121
C5B	0.088	0.12
C6B	0.103	0.112

TABLE 42. APCHE DELAY IN MS

BIBLIOGRAPHY

[1] G. Marfia, C. E. Palazzi, G. Pau, M. Gerla, and M. Roccetti, “TCP Libra: Derivation, Analysis and Comparison with Other RTT-Fair TCPs”, Computer Networks, Elsevier, vol. 54, no. 14, October 2010, pp. 2327-2344.

[2] G. Marfia and M. Roccetti, “TCP At Last: Reconsidering TCP's Role for Wireless Entertainment Centers at Home”, IEEE Transactions on Consumer Electronics, IEEE Consumer Electronics Society, vol. 56, no. 4, November 2010, pp. 2233-2240.

[3] M. Allman, V. Paxon, and W. Stevens, “TCP Congestion Control”, RFC 2581, April 1999.

[4] K. Tan, J. Song, Q. Zhang, and M. Sridharan, “A Compound TCP Approach for High-Speed and long Distance Networks”, INFOCOM 2006, Barcelona, Spain, Apr. 2006.

[5] K. Tan, J. Song, M. Sridharan, and C. Ho, “CTCP: Improving TCP-Friendliness Over Low-Buffered Network Links”, Microsoft Technical Report.

[6] I. Rhee and L. Xu, “Cubic: A New TCP-Friendly High-Speed TCP Variant,” ACM SIGOPS Operating System Review, Volume 42, Issue 5, July 2008, pp. 64-74, 2008.

[7] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, “Host-to-Host Congestion Control for TCP”, IEEE Communications Surveys and Tutorials, vol. 12, no. 3, pp. 304-342, 2010.

[8] H. Jamal and K. Sultan, “Performance Analysis of TCP Congestion Control Algorithms”, International Journal of Computers and Communications, Issue 1, Volume 2, 2008.

[9] J. Chicco, D. Collange, and A. Blanc, “Simulation Study of New TCP Variants”, IEEE symposium on Computers and Communications, June 2010.

[10] M. Bateman and S. Bhatti, “TCP testing: How well does ns2 match reality?”, IEEE AINA, April 2010.

[11] Y. Li, D. Leith, and R. Shorten, “Experimental Evaluation of TCP Protocols for High-Speed Networks”, Technical Report, Hamilton Institute, 2005.

[12] S. Ha, Y. Kim, L. Le, I. Rhee, and L. Xu, “A Step toward Realistic Performance Evaluation of High-Speed TCP Variants”, PFLDnet, Nara, Japan, 2006

[13] V. Paxon, M. Allman, J. Chu, and M. Sargent, “Computing TCP’s Retransmission Timer,” RFC 6298, Jun. 2011.

[14] L. He , R. K. Karne ,A. Wijesinha , and A. Emdadi, “Design and Performance of a Bare PC Web Server,” IJCA, vol. 15, no. 2, pp. 110-112, Jun. 2008.

[15] N. Seddigh and M. Devetsikiotis, “Studies of TCP’s Retransmission Timeout Mechanism,” IEEE ICC, vol. 6, pp 1834-1840, Jun. 2001.

[16] I. Psaras and V. Tsaoussidis, “Why TCP timers (still) don’t work well,” Elsevier Computer Networks Journal, COMNET 2007.

[17] I. Psaras and V. Tsaoussidis, “The TCP minimum RTO revisited,” in Proc. IFIP Networking, Atlanta, GA, May 2007.

[18] A. Gurtov and R. Ludwig, “The Eifel response algorithm for TCP,” RFC 4015, Feb. 2005.

[19] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. Ganger, G. A. Gibson, and B. Mueller, “Safe and effective fine-grained TCP retransmissions for datcenter communication,” in Proc. ACM SIGCOMM, Barcelona, Spain, Aug. 2009.

[20] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay, “Variability in TCP round-trip times,” in Proc. ACM SIGCOMM, Miami, FA, Oct. 2003.

[21] P. Sarolahti, M. Kojo, and K. Raatikainen, “F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts,” in Proc. ACM SIGCOMM, Karlsruhe, Germany, Aug. 2003

[22] Y, Wang, S. Ye, and X. Li, “Understanding Current IPv6 Performance: A Measurement Study,” 10th IEEE Symposium on Computers and Communications (ISCC 2005), pp. 71-76, 2005.

[23] X. Zhou, M. Jacobsson, H. Uijterwaal, and P. Van Mieghem, “IPv6 delay and loss performance evolution,” International Journal of Communication Systems, vol. 21, no. 6, pp. 643–663, 2008.

[24] Md. T. Aziz and M. S. Islam, “Performance Evaluation of Real–Time Applications over DiffServ/MPLS in IPv4/IPv6 Networks,” Masters Thesis, Blekinge Institute of Technology, 2011.

[25] D. T. Ustundag, “Comparative routing performance analysis of IPv4 and IPv6,” Master’s Thesis, Atilim University, 2009.

[26] http://acikarsiv.atilim.edu.tr/browse/100/301.pdf, accessed: July 22, 2012.

[27] M. Nikkhah, R. Guerin, Y. Lee, and R. Woundy, “Assessing IPv6 through web access a measurement study and its findings,” 7th Conference on Emerging Networking Experiments and Technologies (CoNEXT ‘11), pp. 1-12, 2011.

[28] S. Narayan, P. Shang, and N. Fan, “Performance Evaluation of IPv4 and IPv6 on Windows Vista and Linux Ubuntu,” International Conference on Networks Security, Wireless Communications and Trusted Computing, (NSWCTC '09), pp. 653-656, 2009.

[29] A. Berger, “Comparison of performance over IPv6 versus IPv4,” Internet Statistics and Metrics Analysis (ISMA 2012) AIMS-4 Workshop on Active Internet Measurements, pp. 1-30, 2012. http://www.caida.org/workshops/isma/1202/slides/aims1202_acox_supplement.pdf, accessed: July 22, 2012.

[30] C. Ciflikli, A. Gezer, and A. T. Ozsahin, “Packet traffic features of IPv6 and IPv4 protocol traffic,” Turk. J. Elec. Eng. & Comp. Sci., vol. 20, no. 5, pp. 727-749, 2012.

[31] R. K. Karne, K.V. Jaganathan, T. Ahmed, and N. Rosa, “DOSC: Dispersed Operating System Computing”, 20th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA ’05 Onward Track), pp. 55-61, 2005.

[32] R. K. Karne, K.V. Jaganathan, and T. Ahmed. “How to run C++ applications on a bare PC”, 6th ACIS International Conference on Software Engineering, Networking, Parallel/Distributed Computing (SNPD 2005), pp. 50-55, 2005

[33] Multi-Generator (MGEN), http://cs.itd.nrl.navy.mil/work/mgen, accessed: July 22, 2012.

[34] http_load, available at http://www.acme.com, accessed: March 13, 2012

[35] Wireshark packet analyzer, http://www.wireshark.org/, accessed: July 22, 2012.

CURRICULUM VITA

NAME: Alae Loukili

PERMANENT ADDRESS: 696 Massira I, Marrakech, Morocco

PROGRAM OF STUDY: Information Technology

DEGREE AND DATE TO BE CONFERRED: Doctor of Science, December 2012

SECONDARY EDUCATION: Zerktouni High School, Morocco

Universities Attended:

1. Towson University , Jan 2008- Dec 2012., DSc Information Technology
2. Towson University, Aug 2002- May 2004, MSc Computer Sciences
3. ENSEM, Casablanca-Morocco, Aug 1994- Aug 1998, State Engineer Electrical Engineering
4. Cadi Ayyad University, Marrakech-Morocco, Sep 1991 – May1994, Associate Math-Physics

Professional publications:

1. **A. Loukili**, A. Wijesinha, R. Karne and A. Tsetse, "Web Server Performance with Cubic and Compound TCP", IASTED International Conference on Communication, Internet, and Information Technology (CIIT2012)
2. **A. Loukili**, A. Wijesinha, R. Karne and A. Tsetse, "TCP's Retransmission Timer and the Minimum RTO", 6th International Workshop on Performance Modeling and Evaluation of Computer and Telecommunication(PMECT 2012)
3. **A. Loukili**, A. L. Wijesinha, R. K. Karne, and A. K. Tsetse, "Performance of an IPv6 Web Server under Congestion," 12th International Conference on Networks (ICN 2013)
4. A. K. Tsetse, A. L. Wijesinha, R. K. Karne and **A. Loukili** , " A Bare PC NAT Box", International Conference on Communications and Information Technology (ICCIT 2012)
5. A. K. Tsetse, A. L. Wijesinha, R. K. Karne and **A. Loukili** , "A 6to4 Gateway with Co-located NAT", IEEE International Conference on Electro Information Technology (EIT 2012)
6. A. K. Tsetse**,** A. L. Wijesinha, R. K. Karne and **A. Loukili** , "Measuring the IPv4-IPV6 IVI Translation Overhead", ACM Research in Applied Computation Symposium(RACS 2012)