An Empirical Study

of TCP’s Fairness

Abstract of dissertation entitled:

An Empirical Study of TCP's Fairness

submitted by Ng Ping Wing

for MSc in Information Technology

at The Hong Kong Polytechnic University in April 1999.

It has been reported that TCP connection discriminates against long round trip times with regards to the throughput. The cause of the unfairness is the "slow start" and "congestion avoidance" algorithms used by TCP, which result in unfair bandwidth allocation when multiple connections with different round trip times share a congested gateway. This fairness problem is particularly severe for networks with long round trip delay. An experimental approach is used in this project to identify this fairness problem. To solve this problem, two algorithms have been proposed by some researchers. These algorithms, termed as the "linear rate policy" and "constant rate policy", have been implemented and studied. The result is that these algorithms solve the problem to a certain extent but with some shortcomings. During the experiment, a new factor is discovered based on the measurement data. This finding reveals that there is a positive correlation between the round trip time and the congestion window during the congestion phase. Based on this finding, a new algorithm is then devised and implemented to address this problem together with experimental measurements of how well this algorithm performs. Judging from the measured result, it is found that the "linear rate policy" and "constant rate policy" suffer from a fundamental disadvantage of inducing more congestion during the congestion phase. Hence

additional insight about the nature of congestion is gained from the measurements. The new algorithm is found to be very promising because it do not suffer from the mentioned fundamental disadvantage in addition to its capability to restore fairness. This sheds light on future improvement and implementation of TCP, and there is plenty of rooms for future work.

Acknowledgements

To work on a MSc degree in part time mode is a difficult task, especially for one who is heavily loaded by daily jobs like me. Anyway, this thesis comes out now and I can relax for a while. This will be good news for my boys because their father can play with them in the evening again which they have been expecting for a few years.

This thesis is about TCP, an end-to-end transport protocol used in the current Internet, so there are lots of terms like “congestion” and “acknowledgement” etc. Congestion, it is a situation of heavy loading at which I encountered frequently. In TCP, acknowledgement needs something to trigger it. There are many, for examples, ideas and materials in this MSc thesis originated from my supervisor, the curiosity and my interest in networking, the courtesy for limited available squeezed time from my boss, and so the acknowledgements begin:

I thank my supervisor, Dr. Rocky Chang, for his valuable ideas, comments and help. Without him, this thesis would be in the pending queue of the Hong Kong Polytechnic University’s (HKPU) MSc student list. **When timeout, it would be dropped forever.** Moreover,

I thank myself for hard working.

I thank my wife for her support and understanding.

I thank my boys for their patience.

I thank my boss for his kindness and support.

Finally, I thank all my friends and colleagues who have contributed their time for discussion and ideas.

Thank you all, this project is finished. May be another one is coming ……

Contents

Chapter I Introduction

This thesis studies the behaviour of Transmission Control Protocol (TCP), especially addressing the issue of fairness at a congested gateway situation under which there is a discrimination against long round trip times (RTT). This thesis will explain the fairness problem, the methodology used to examine the problem and the experimental set-up used to take measurements. Finally, the long round trip times issue is addressed by using an adaptive scheme, which modifies the TCP window-increase algorithm and the experimental measurement results presented. The result of this scheme towards achieving the fairness goal for a TCP network and possible future work is discussed in the conclusions.

1.1 Motivation

Nowadays, the Internet appears to achieve a great success in promoting information exchange and electronic commerce. Everyone talks about Internet and exciting applications that come out nearly every day. However few cares about the underlying infra-structure that supports the Internet, namely a global network that operates under the Transmission Control Protocol and Internet Protocol (TCP/IP) suite. All will suffer if there is any performance problems with that infra-structure. Also, when everybody is on the Internet, the fairness issue certainly will attract peoples' attention.

There are many different viewpoints on how fairness should be defined, such as based on throughput or should each connection receive the same amount of allocation on the possible congested resources. In reality there exists unfair situation in TCP/IP networks with congested gateway with respect to throughput. From a user point of view, the poor throughput and hence unbearable long waiting time while downloading a large file from a remote site via a modem line through a congested gateway is a concrete example of this fairness issue. Other example such as interrupted responses while working with telnet at a remote site through a congested gateway is frequently encountered. With the developments in ultra-high speed networks and applications that use TCP/IP, the fairness issue will amplify and surface. Therefore this issue should be tackled and appropriate solution be built into the TCP/IP implementations promptly.

1.1 Objective

TCP/IP networks discriminates against long round trip times with connections passing through multiple congested gateways. The cause of the fairness problem stems from the “slow start” and “congestion avoidance” algorithm used by TCP, which result in unfair bandwidth allocation when multiple connections with different round trip times share a congested gateway. This happens from time to time in real world scenarios.

This project uses a testbed for measurements and experiments with modified window increment schemes. The goal is to improve fairness for TCP/IP network under a

congested situation. New algorithms will be implemented and evaluated. Using a testbed network to study and rectify the problem is of practical significance since the fairness problem is of common occurrence in the real world. By actually connecting the machines together under a practical working environment, the problem can be studied without overlooking factors that are of prime importance.

1.2 Organization of This Thesis

Chapter 2 summarizes TCP features and mechanisms that are relevant to this thesis. It also briefly describes previous work related to the fairness issue. In chapter 3, the experimental set-up will be described and how it can be used to study the problem is explained together with some measured data. Subsequently, in chapter 4 the modified algorithm will be presented together with the measured result. Chapter 5 describes miscellaneous issues such as gateway effect, bursty traffic, phase effect and observed dynamic round trip times variation in real cases. Chapter 6 proposes a new algorithm and describes its implementation. The test results are then presented. Finally, chapter 7 concludes this work and discusses possible future work.

1.3 Contribution

The main contributions of this project are summarized in the following:

This project involves real measurement and verification of this current topic. All previous results are based on simulation study and observation only. It is the author's view that although simulation is a very important tool for studies and researches; its validity depends on the exact and faithful implementation of the simulator. For example, the protocol handling routines may not be the actual coding embedded in the operating platforms. Also the mathematical model used by the simulator may not be appropriate for the situation. For example, Poisson process may be assumed for certain types of traffic, which may not be the case in the actual traffic. The actual environment being modeled may have many factors not considered by the simulator during the simulation exercise. For example, variables may be treated as invariants during the simulation exercise for simplicity. Therefore, experimental proof is absolutely necessary and valuable. However, due to the network's complexity, simulation still plays a vital role in attempting to characterize how different facets of the network behave, and how proposed changes might affect the network's different properties. But the essential point here is, measurement is absolutely necessary for a crucial reality check. It also serves to test those assumptions used in simulations.

1. The positive correlation between round trip time and congestion window during congestion is discovered. That is, the round trip time increases as the congestion window increases during congestion avoidance phase. This has not been mentioned in any previous papers. This factor correlates with the traffic pattern and can be used as a measure for predicting the likelihood of congestion. Monitoring the rtt variation probes the likelihood of congestion in real time. Taking this factor into account during analysis provides additional insight into the fairness problem.

2. The criteria for a new algorithm to solve the fairness problem are proposed, taking into account the findings in this project. Then a trial algorithm is implemented and tested. Further insight on the fairness problem is gained with the test result of this trial algorithm. This algorithm seems to be very promising for future TCP implementations, although its scaling capability is not fully investigated.
3. During the experiment, it is discovered that extreme care must be exercised for topological consideration such as where the bottleneck gateway is located. It is a practical issue when doing experiment. In brief, topological consideration in real scenario bears significance when analyzing data for the fairness problem.

Chapter 2 TCP and the fairness problem

TCP provides a reliable stream service with the ability for a full duplex connection. TCP uses acknowledgement and sliding window for flow control. It also features congestion avoidance algorithm and heuristics implementation to avoid commonly encountered problems such as congestion and deadlock. It is a well-tested networking protocol now supporting the operation of the global Internet. On the other hand, it has its own short-comings. One of these is the fairness problem when connections with different round trip times going through a shared congested gateway. In this chapter, we shall provide a brief overview of important ideas embedded in TCP and how the problem of fairness arises. Finally, this chapter is concluded with related works on the problem of fairness.

2.1 Brief Overview of important ideas in TCP

As quoted by Comer[3], “TCP is a communication protocol, not a piece of software”. It is important to understand that implementation of a particular protocol doesn’t always reflect what the protocol means or what it wants. The relationship between the definition of a programming language and a compiler is a good analogy.

TCP is quite a complex protocol, details of it can be found in the Request For Comments (RFC) posted by Postel [5]. Simply speaking, TCP is a reliable connection-

oriented stream transport protocol, but it assumes very little about the underlying communication system. Most common implementations make it to run on top of the Internet Protocol (IP) and form the universally known TCP/IP protocol suite. It features acknowledged connection initiation and termination, in-order and unduplicated delivery of data by using sequence number as data packet label, acknowledgement and sliding window for flow control, retransmission of lost or un-acknowledged data packets which provides the underlying reliability, check-sum for error control, congestion control for busy network and out-of-band indication of urgent data. Concepts related to the fairness problem either directly or indirectly are briefly described below.

2.1.1 Sliding Window and Flow Control

The objective of TCP's sliding window mechanism is two fold: flow control and high throughput. The TCP window mechanism allows the sender to send multiple segments (segment is the unit of a data packet in TCP) before an acknowledgement arrives. It also restricts the receiver's transmission until it has sufficient buffer space to accommodate more data for end-to-end flow control. The TCP sliding window operates at the octet level. Octets of the data stream are numbered sequentially (sequence number) and the sender keeps three pointers for each connection to maintain the sliding window. By doing so, it allows the window size to vary over time. The greatest advantage of using a variable window is that it provides reliable transfer in addition to flow control because the receiver can tell the sender about its buffer availability via the window advertisement. This is important in the highly "heterogeneous internet environment", where machines of various speeds and

processing power communicate through networks and routers of various speeds and processing power. Basically, TCP solves the end-to-end flow control problem but when intermediate routers become overloaded (a congested condition), it does not have any explicit methods to rectify the condition. In theory TCP can do nothing to the congested routers, but only depends on the congestion avoidance algorithm (described later) for remedy.

2.1.2 Acknowledgements

In TCP, acknowledgement associated with a sequence number refers to a position in the bytes stream under transmission. As data packets can be lost or delivered out of order, the receiver uses these sequence numbers to re-assemble the data. In order to have an ordered construction of the data, the receiver always acknowledges (piggy-backing is used) the longest contiguous prefix of the stream that has been received correctly. Each acknowledgement specifies a sequence number of the next octet that the receiver expects to receive. This is a cumulative scheme because it reflects how much of the stream has been received.

2.1.3 Timeout and Retransmission

Starting from the very beginning, TCP is designed for an internet environment. Because of this consideration, the underlying data link transmission mechanism will have data lost and corruption is assumed. Also, in such a global network, the rate of acknowledgements coming back will vary greatly and be in a dynamic state. Therefore whenever TCP sends a data segment, it starts a timer and waits for an

acknowledgement. If this timer expires before the acknowledgement comes back, TCP has no way but assumes that the segment was lost or corrupted. It has to retransmit the segment for reliability. To accommodate the varying delays encountered in an internet environment, TCP uses a retransmission algorithm that monitors delays on each connection and adjusts the corresponding timeout parameters accordingly. For this algorithm to work, TCP has to record the time at which each segment is transmitted and the time at which the acknowledgement comes back. From the two records, TCP computes the time difference and it is known as the round trip time(rtt). The accuracy of this rtt and the corresponding derived parameters are of importance for TCP to perform satisfactory.

2.1.4 Measurement of Round Trip Times

In a heterogeneous internet environment, variation in round trip time (rtt) is an inevitable matter. Traditional TCP software stores the estimated rtt as a weighted average and uses samples from new rtt measurements to change the average slowly. However, complications arise because TCP uses a cumulative acknowledgement scheme in which an acknowledgement refers to data received, and not to the instance of a specific segment that carried the data. Due to the retransmission mechanism implemented for reliability and this cumulative acknowledgement scheme, the sender has no way to know in advance whether an acknowledgement refers to the original or retransmitted segment. This phenomenon is known as the “acknowledgement ambiguity”. Because the expiration timer for retransmission depends on rtt and the acknowledgement ambiguity’s problem, measurement of rtt is not a trivial matter.

To solve the captioned difficulties, Karn's algorithm stated that when computing the round trip estimate, samples corresponding to retransmitted segments should be ignored; also a backoff strategy should be used for timeout. The backoff technique computes an initial timeout using an accepted formula. However, if the timer expires and causes a retransmission, TCP increases the timeout. Most current implementations double the timeout (binary backoff). When an internet misbehaves, Karn's algorithm avoids the difficulties of timeout calculation by computing timeout value not from the current round trip estimate. Actually, it computes an initial timeout using current rtt but then backs off the timeout value on each retransmission encountered until it can successfully transfer a segment. When it sends the following segments, it retains the timeout that is resulted from backoff. Eventually, when an acknowledgement corresponding to a segment that does not require retransmission arrives, TCP then recomputes the rtt and resets the timeout value accordingly. Real world experience [3] shows that Karn's algorithm works well even under a high packet loss network.

In order to further improve TCP performance, the 1989 specification for TCP requires implementations to estimate both the average rtt and the variance. As a result, new implementations of TCP can adapt a wider range of variation in delay and subsequently yield a higher throughput. The new specification can be summarized by the following equations for rtt and timeout calculation:

$$DIFF = SAMPLE - old\_rtt$$

$$smoothed\_rtt = old\_rtt + \delta * DIFF$$

$$DEV = old\_DEV + \rho(|DIFF| - old\_DEV)$$

$$Timeout = smoothed\_rtt + \eta * DEV$$

where DEV is the estimated mean deviation, δ is a fraction between 0 and 1 that controls how quickly the new sample affects the weighted average. ρ is a fraction between 0 and 1 that controls how quickly the sample affects the mean deviation, and finally η is a factor that controls how much the deviation affects the round trip timeout. Research suggests values of $\delta = 1/2^3$, $\rho = 1/2^2$, and $\eta = 3$ is a good choice. The original value for η in 4.3 BSD Unix was 2; it was changed to 4 in 4.4 BSD Unix.

2.1.5 Slow Start and Congestion Avoidance

TCP is designed to react to congestion caused by intermediate gateways. When congestion occurs, it means that the round trip delay will increase due to overloading of the gateway and segments may be dropped by the gateways. TCP as an endpoint does not know the details of where congestion has occurred or why. It can only react by doing retransmissions which will further congest the gateways, resulting in "congestion collapse". To avoid congestion collapse, TCP will reduce transmission rates when congestion occurs; the mechanism employed is "slow start and congestion avoidance". It works as follows:

For each connection, TCP remembers the receiver's window and maintains a congestion window. The window actually used is the minimum of the above two windows. In steady state, the congestion window should be the same as the receiver's

window. Reducing the congestion window reduces the TCP's traffic injected into the network. Upon loss of a segment, TCP reduces the congestion window by half with a minimum of one in case of multiple reductions. And for those segments that remain in the allowed window, they will backoff the retransmission timer exponentially. The result is that TCP reduces the volume of traffic and rate of transmission exponentially hoping that intermediate gateways will recover during this period.

When congestion ends, TCP uses the "slow-start" technique to scale up transmission. Basically, whenever TCP starts a new connection or recovers from congestion, it starts the congestion window as one and increases the congestion window by one each time an acknowledgement arrives. This actually is an exponential increment again but TCP will restrict the congestion window to one-half of its original value before congestion during this phase and then enter the so called "congestion avoidance" phase during which its rate of increment will slow down. During congestion avoidance, TCP will increase the congestion window by one only if all the segments in the window have get acknowledged.

It is worthwhile to note here that most implementations of TCP maintain the congestion window in size of bytes and have incorrectly add a term, (maximum segment size / 8), to the congestion avoidance increment algorithm. This term should be left out in future implementation of TCP as pointed out by S.Floyd, G.R.Wright and W.R.Stevens [see 7]. This is a problematic term which has been proved by various simulation results.

To summarize, TCP uses slow start and congestion avoidance in addition to measurements of variation of rtt and exponential timer backoff to improve the performance. Versions of TCP that uses these techniques have improved the performance of previous versions by factors of 2 to 10.

2.1.6 The Self-Clocking Mechanism

The self-clocking mechanism of TCP is essential to its basic operation and performance. Basically, this mechanism can be explained as follows:

> Whenever a TCP receiver sends an acknowledgement, this acknowledgement will trigger new data segments from the sender when it is received properly, because these acknowledgements advance the window. Thus, the TCP's acknowledgements can be seen as a kind of "clock signal" to strobe new data segments into the network. Hence, the protocol by itself is "self-clocking" because the acknowledgements can be generated no faster than the data segments can get through the network.

The self-clocking mechanism is at the center of Jacobson's idea of "conservation of packets" [6]. If a connection is running steady with a full window of data in transit, it is said to be in equilibrium. A connection in equilibrium is "conservative" if a new packet is not injected into the network until an old packet leaves. The self-clocking mechanism indicates that an acknowledgement signals a packet has been received or in other words has left the network. Therefore it is a good indication that a new packet can be injected into the network. In fact, this mechanism can be exploited for

congestion avoidance since data packets arrive at the receiver can never faster than the limit as restricted by the bottleneck link bandwidth. If the receiver's acknowledgements arrive at the sender with the same spacing, so in theory the sender by sending new data packets at the same rate can avoid overloading the bottleneck link. However the self-clocking mechanism is subjected to distortion by acknowledgements that do not arrive with the same spacing. For example, when response to congestion, the congestion avoidance scheme will restrict the sender from sending more segments into the network to avoid further congestion. As a result, acknowledgements stop coming back and so no new segments can be strobed out. The connection is said to be temporarily dried up. Such dry spells are damaging to TCP's performance.

The challenge for TCP during the initial stage is to attain equilibrium as far as possible without overloading the network. With very little information about the network, the start-up algorithm plays an important role in how to achieve this objective. The current practice is the "slow start and congestion avoidance" scheme as mentioned in 2.1.5.

2.2 The Fairness Problem

The congestion avoidance algorithm used by TCP results in unfair bandwidth allocation when multiple connections with different round trip times share a congested gateway.

The problem stems from the algorithm. After the initial slow start phase of window growth, TCP effectively attempts the same fixed change to the usable window size. The sending rate of a connection therefore increased bit by bit for every rtt interval, regardless of the value of the rtt. The algorithm make TCP probes for available bandwidth in the "linear" growth phase of congestion avoidance by increasing the send window size by a single segment every rtt. Under this scheme, long rtt connections increase their effective injection rate more slowly than the short rtt connections, leading to unequal allocations of bandwidth on the link shared by these flows. Since retransmission timers scale with the rtt, timeouts hurt longer rtt connections more than shorter rtt connections. Short rtt connections will be able to retransmit, see the effects of the retransmission, and enter linear growth faster, allowing them to recover bandwidth even before the longer rtt connection retransmits.

2.3 Related Work

The original concept for the additive increase, multiplicative decrease policy in TCP comes from DECnet which was originated from Jain and colleagues [9]. While connections with different rtts are not considered in this paper, the authors show that the captioned policy will converge to fairness.

Several researchers observed and simulated the biased effect in TCP for connections with long rtts. In a paper by Sally Floyd and Van Jacobson [1], they concluded that

the rtt bias in TCP/IP networks was resulted from the TCP window increase algorithm and not related to the gateway dropping policy. They briefly discussed how changes to the window increase algorithm could eliminate this bias. The work by Mankin [10] also identifies the bias against long delay connections.

In another paper by Sally Floyd [2], she developed a heuristic analysis for the fairness problem and showed that eliminating the bias against longer rtt connections is sufficient to achieve acceptable fairness. Simulation had been done for some simple topologies to illustrate the problem. In fact, Floyd developed a fairly general characterization of the window increase algorithms to address the problem. A key assumption is that a number of packets approximately equal to the send window size are sent every rtt. If c is a constant (in the simulation, c=4 is used) subject to different simulation environment, then the window should grow at a rate of c/rtt packets per rtt, which is c/rtt^2 packets per second per second. This algorithm is being termed as the "constant rate increase policy" [2] or "constant rate", while the increase rate with the characteristic of c/rtt is being termed as the "linear increase rate policy" or "linear rate". Further simulations have been done by a research group at U.C.Berkeley. Their findings showed that the "constant rate" when applied globally in their simulation network, could often (but not always) achieve simultaneous high utilization and high fairness. However, when selectively applied, the performance was sensitive to the correct tuning of the rate constant c, with the result that an over-aggressive connection could in fact hurt itself. On the other hand, a less aggressive "linear rate" policy limits the fairness that could be achievable. Anyway, it did offer significant gains in many cases and performance was less sensitive to the proper selection of the constant c.

Work related to the fairness issue for TCP for multicast has been done by Huayan Amy Wang and Mischa Schwartz. They proposed a window-based "Random Listening Algorithm" [8]. The nature of the problem is essentially the same but they address mainly the multicast issue.

Also in a paper by T.Lakshman and U.Madhow [11], they made the observation that TCP is grossly unfair towards connections with higher round trip delays and suggested that an alternate dynamic window mechanism is a high priority for future research, although they do not endorse any new mechanism.

The work in this thesis differs from all the previous in that it is an experimental verification of the problem rather than through simulations. The "linear rate" and "constant rate" policies have been implemented in a Linux kernel and tested. Also the positive correlation between the variation of rtts and the congestion window during congestion phase has been discovered. This sheds light on an even better dynamic window adjustment algorithm for this problem. Finally, the criteria and properties of a new possible algorithm are proposed and discussed. A trial implementation is then tested and the results concluded.

Chapter 3 Empirical Study on a Testbed Network

The experimental set-up for investigating the TCP's fairness problem should be simple enough so that measurements could be taken easily while there should be no sacrifice of reality. The experimental set-up and design should also allow flexibility for modification so that different scenarios could be tested. Details for the set-up are described in the following sections.

3.1 Methodology

The measurement is taken using the program "tcpdump", which is a shareware available on many Unix platforms. Data about overall throughput is reported by the ftp (an application protocol) applications. In addition a small Korn shell script is developed to calculate the throughput based on those tcpdump output files for different period during the measurement. A PC running linux (mainly a PC based Unix Operating System) acts as the TCP source under investigation. Linux is used because it has all the source codes available and its kernel can be modified to suit the experimental need. In fact, the TCP codes of the Linux kernel have been modified in order to dump some kernel variables for examination. Also the modified window increase algorithms are implemented on the Linux kernel under experimentation. Congestion states are monitored by looking at the statistical report of the router and

the dumped out kernel variables. Power PCs running AIX (a commercial Unix Operating System developed by IBM) are used as the clients, AIX features a “ping” utility that can control the packet size and a command switch for flooding the network. This feature is used to congest the gateway under investigation in the current experimental set-up. More detailed description will be provided in the following sections.

3.2 Experimental Set-up

Diagram 3.1 shows the experimental set-up in details. Some equipment such as networking hubs are omitted from the diagram in order to focus on the main networking components. It is believed that it will not affect the experimental details, results, and the generality of the experiment. The current set-up is based on Ethernet technology running 10Mbps on cat. 5 UTP cables.

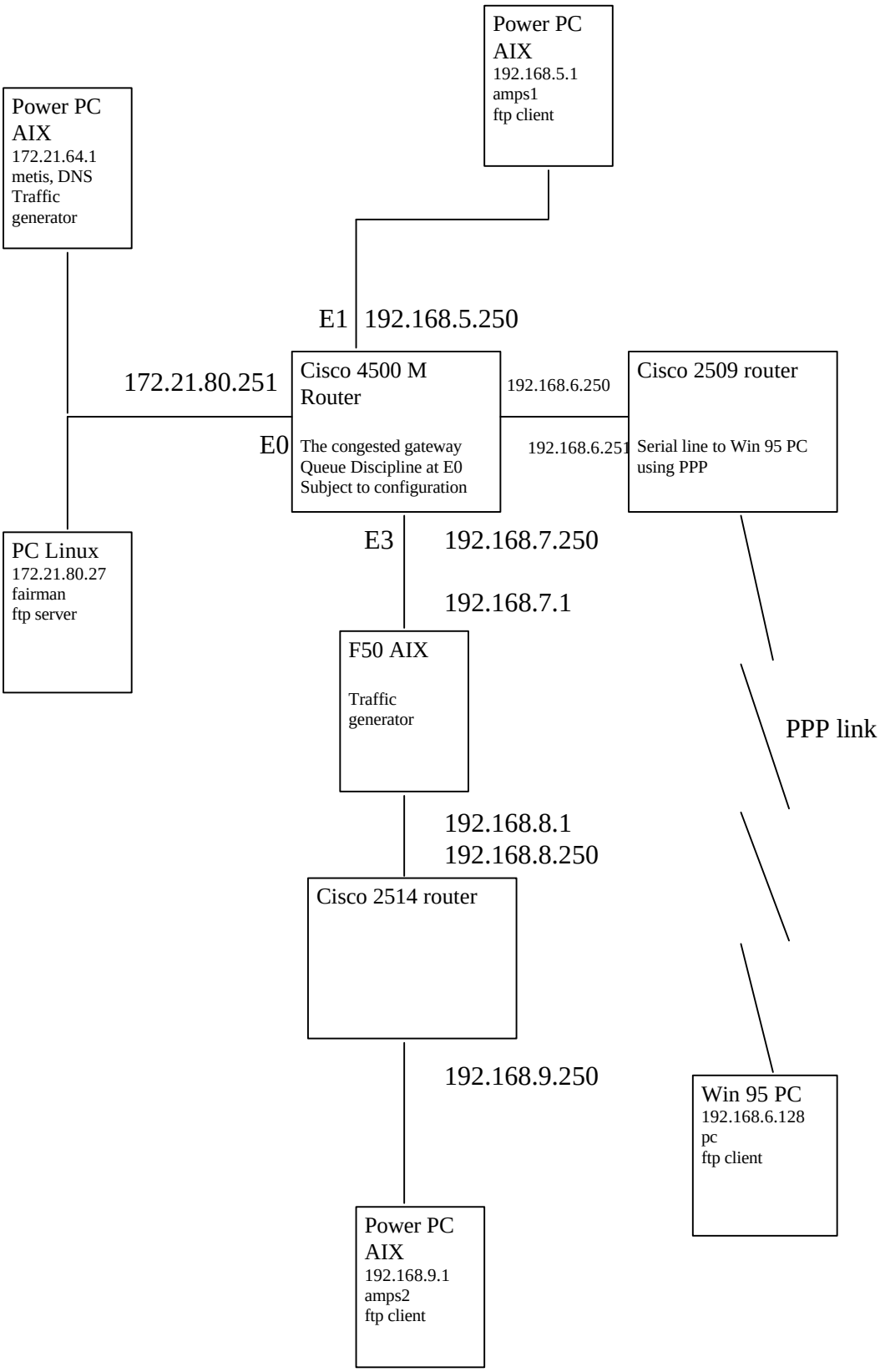

Power PC
AIX
192.168.5.1
amps1
ftp client
Power PC
AIX
172.21.64.1
metis, DNS
Traffic
generator
E1
192.168.5.250
172.21.80.251
Cisco 4500 M
Router
192.168.6.250
Cisco 2509 router
E0
The congested gateway
Queue Discipline at E0
Subject to configuration
192.168.6.251
Serial line to Win 95 PC
using PPP
PC Linux
172.21.80.27
fairman
ftp server
E3
192.168.7.250
192.168.7.1
F50 AIX
Traffic
generator
PPP link
192.168.8.1
192.168.8.250
Cisco 2514 router
192.168.9.250
Win 95 PC
192.168.6.128
pc
ftp client
Power PC
AIX
192.168.9.1
amps2
ftp client

Fig. 3.1 Diagram showing the experimental set-up for fairness measurements 3.2.1

3.2.2 Gateways (routers) set-up

Cicso routers were used in this experiment. They run IOS (the operation system for Cisco router) version 10.3 or beyond. Queuing discipline in use is “First In First Out” (FIFO) unless otherwise specified. In the gateway effect investigation experiment, the discipline was configured to use “Random Early Detection” (RED). The routing protocol used is Routing Information Protocol (RIP). Cisco router features command set which allow users to view various statistics and information. These have been used to monitor the status of the Cisco 4500M to ensure that this designated bottleneck gateway is dropping packets during the experiment.

3.2.3 PC Linux set-up

The PC Linux was installed from the RedHat package running kernel level 2.0.27. Its TCP/IP features delayed acknowledgement, large window support, fast recovery and retransmission and fine granularity rtt. The kernel was in fact modified to dump the variables: destination address, mss, ssthresh, cwnd and rtt for monitoring purposes. It should by no mean affect its operation except a little bit downgrade of performance, which is not the essential point of this experiment. The TCP routine was later modified to use the “linear rate”, “constant rate” and some new algorithms for experimentation. Since the main point in this experiment is a one-way traffic from the traffic source, the PC Linux in this case, there is no need to make any modification to the client side. The program “tcpdump” was run in this machine to gather the data needed for analysis.

3.2.4 Power PCs set-up

The Power PCs serve two purposes. One is as the ftp clients in this experiment. The other is as a traffic generator, injecting packets into the network to congest the gateway. The former consists of hosts “amps1” and “amps2”, while the later consists of host “metis” which also work as a Domain Name Server (DNS). The AIX version in these machines is 4.2.1 which features delayed acknowledgement, large window support and fast recovery and retransmission.

3.2.5 Win 95 PC set-up

This is a typical Win 95 machine on an IBM thinkpad notebook. The original purpose for this set-up was to test a long rtt PPP link. However, it is later understood that this set-up won’t work because of converging traffic from a large capacity one into a low capacity one generates congestion at the Cisco 2509 router. A sample dump and plot showed this effect at the next section.

3.2.6 F50 set-up

The main purpose of this IBM F50 AIX 4.2.1 machine was as a delay generator in addition to running as a gateway. Two ethernet cards were installed on this machine with IP forwarding enabled inside the kernel. Routing was by means of static routes. Delay in rtt was introduced for the client “amps2” by varying the load on this machine, injecting packets into the network etc. Korn shell scripts were developed on this

machine and run by the “cron” daemon for randomly introducing traffic into the network. The main objective was to avoid the “Traffic Phase Effect” as described in section 5.1.

3.2.7 Measurement

With the above set-up, measurements are taken under five different scenarios for comparison using the application ftp, which is both a tcp server and client, to obtain the throughput:

Case a : no congestion, measurements taken for each client running separately;

Case b : no congestion, measurements taken for both clients running concurrently;

Case c : congestion well before measurements started, both clients running concurrently, lightly congested gateway;

Case d : congestion well before measurements started, both clients running concurrently, heavily congested gateway;

Case e : congestion began with some delay after application started, both clients running concurrently.

The rationales behind the above experimental design is given as follows:
For case a, the objective was to ensure the whole set-up is running properly. For case b, the objective was to ensure the two clients running together won’t introduce any congestion to the gateway. Also these two cases can be used as a reference datum. For cases c and d, they are the scenarios under investigation. For case e, it was used

to check whether the initial window set-up played a vital role during the transfer or not.

As mentioned before, throughputs are reported by the ftp applications and congestion status is achieved by using the ping utility. Randomness is driven by the cron daemon running Korn shell scripts at the F50 machine.

3.3 Result

To demonstrate the fairness problem, difference in rtt is important but it is not the only concern. For example, early in the current experimental design, it is thought that long rtt is the essential factor and hence a PPP link using telephone line as compared to a 10Mbps ethernet should make the rtts contrast a lot. With this idea in mind, a Cisco 2509 router was used to configure the PPP link running at 22.8 Kbps connecting to a Win 95 PC as client. The rtt so connected was approximately 1 ms against 150 ms for "amps1" and "pc" respectively under normal condition. This difference in rtt should trigger the fairness problem between the two connections. Surprisingly, all measurement results did not indicate this is the case. Therefore further analysis was carried out to study it. It was then discovered that one simple concept was overlooked in the design. The set-up made the sender converging traffic from a large capacity network into a low capacity network. This generated congestion at the 2509 router even in trials with the "pc" running as the only client. It should be noted that there was no congestion at the Cisco 4500M router in this case because the traffic generator was

not started. Later when the traffic generator started and congestion introduced to the Cisco 4500M router, the client “amps1” faced congestion and throughput was subsequently reduced for both “amps1” and “pc”. However, congestion at 4500M affected “amps1” more profoundly than “pc” which was always in congested state. The final outcome is throughput dropped significantly for “amps1” as compared to “pc”. The intuition is that it is more unfair to “amps1” as compared to “pc”. The result surely contradicted the expectation if rtt is considered as the only factor for this problem. Figures 3.2, 3.3 and 3.4 illustrated the result.

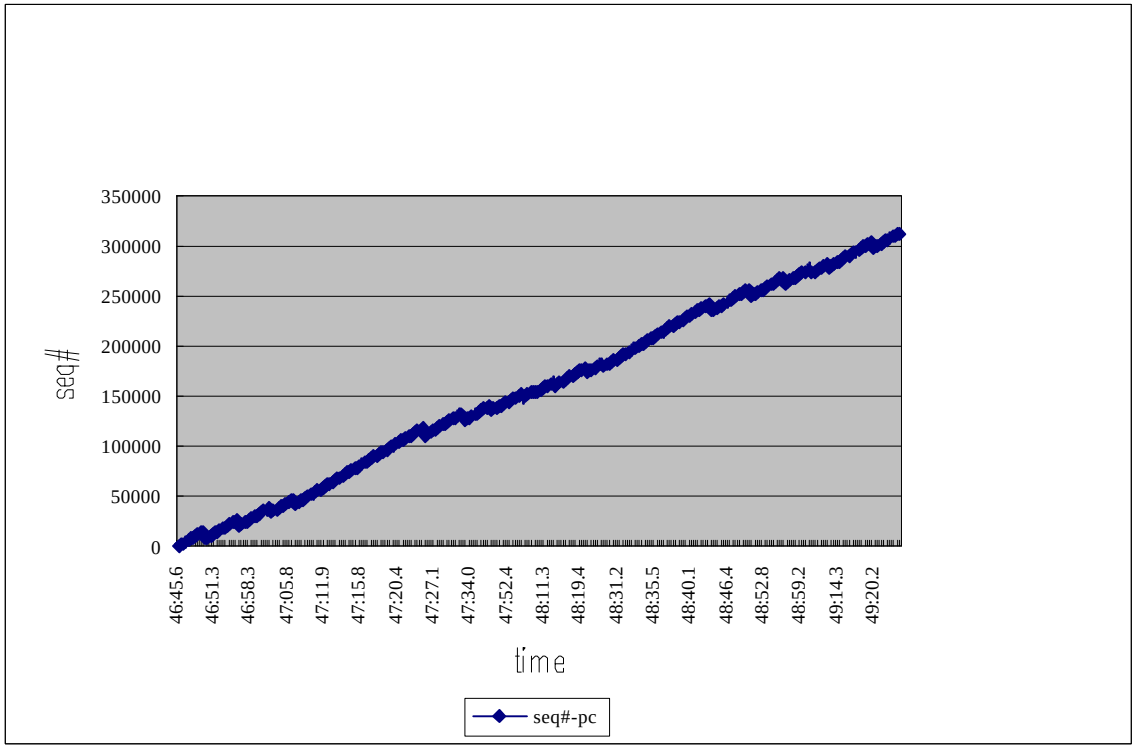


Figure 3.2 Plot of sequence number (seq#) against (vs) time (t) when “pc” was running as the only client

From this figure, one could see that there were many retransmissions even when only the “pc” was running as client illustrating the piping problem. No such retransmission was found when “amps1” was running as the only client.

When both “amps1” and “pc” were running together and put into a congested state, the corresponding plot looked as follows (dark line is for seq# of “amps1”, while grey dots is for seq# of “pc”):

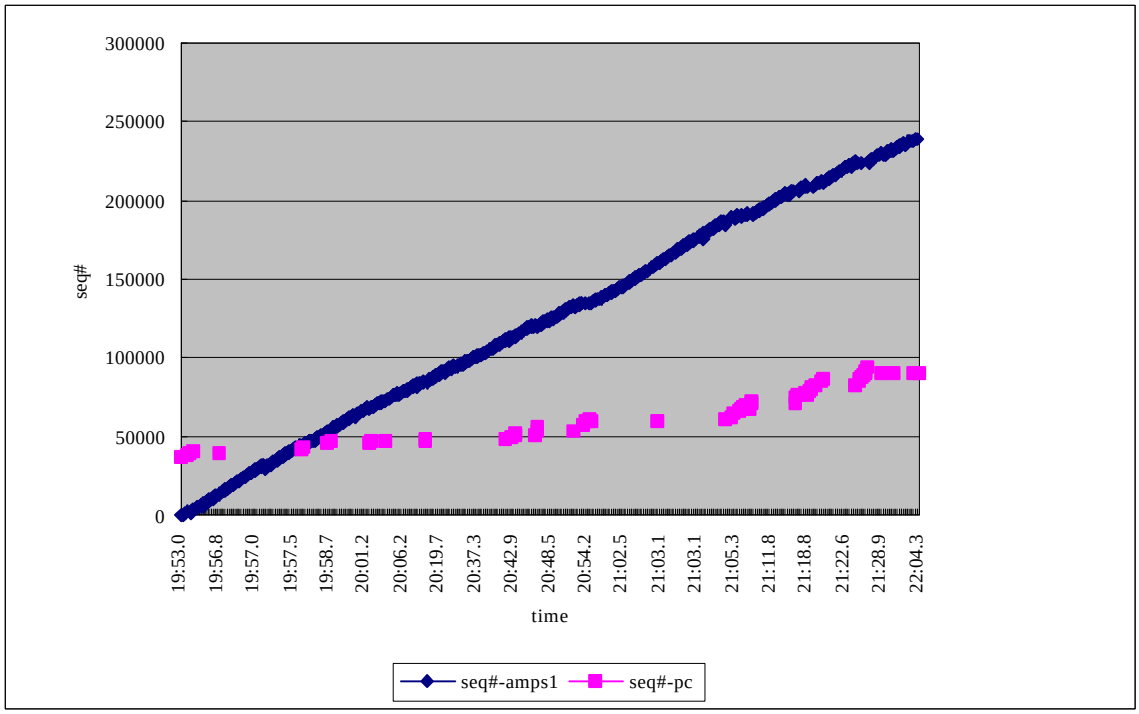


Figure 3.3a Plot of seq# vs t for “amps1” and “pc” when both “amps1” and “pc” were running as clients

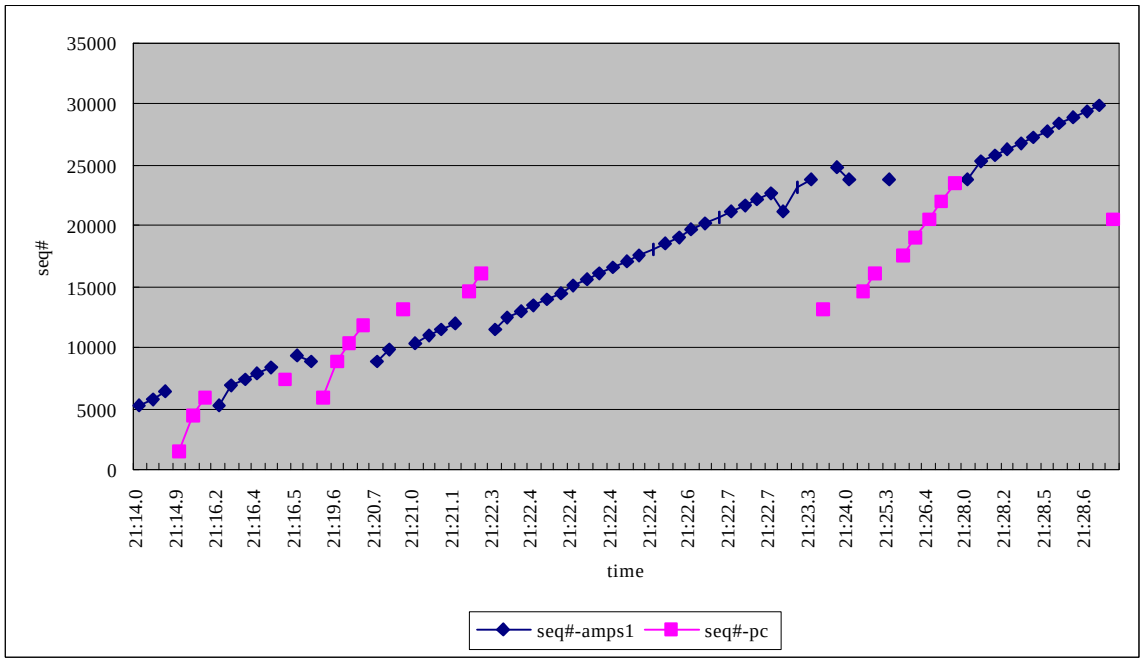


Figure 3.3b Plot of seq# vs t for "amps1" and "pc" when both "amps1" and "pc" were running as clients. Enlarged, offset the y-axis and showing a fraction only.

From the above figures, one can see that both "amps1" and "pc" were doing retransmissions. For "amps1", the spacing between points was much closer than those of "pc", indicating the rtt of "amps1" was shorter than those of "amps2". The seq# axis was re-scaled for plotting them together. Also exponential backoff for "pc" could be deduced from this plot (at time from 21:22.3 to 21:23.3 in fig.3.3b).

The state of both "amps1" and "pc" was checked by the Unix's "netstat" command to ensure the applications were in a correct state. It is listed below for reference:

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:imap                  *:*                     LISTEN
tcp        0      0 *:auth                  *:*                     LISTEN
tcp        0      0 *:pop                   *:*                     LISTEN
tcp        0      0 *:pop-2                 *:*                     LISTEN
tcp        0      0 *:finger                *:*                     LISTEN
tcp        0      0 *:gopher                *:*                     LISTEN
tcp        0      0 *:time                  *:*                     LISTEN
tcp        0      0 *:smtp                  *:*                     LISTEN
tcp        0      0 *:telnet                *:*                     LISTEN
```

```
tcp        0      0 *:ftp                   *:*                     LISTEN
tcp        0      0 fairman:ftp             pc.hko.gov.hk:1028      ESTABLISHED
tcp        0      0 fairman:ftp             amps1.hko.gov.hk.5:1058 ESTABLISHED
tcp        0  36500 fairman:ftp-data        pc.hko.gov.hk:1032      ESTABLISHED
tcp        1      0 fairman:ftp-data        amps1.hko.gov.hk.5:1060 TIME_WAIT
tcp        0      0 *:printer               *:*                     LISTEN
tcp        0      0 *:shell                 *:*                     LISTEN
tcp        0      0 *:login                 *:*                     LISTEN
udp        0      0 *:time                  *:*
udp        0      0 *:ntalk                 *:*
udp        0      0 *:talk                  *:*
udp        0      0 *:syslog                *:*
raw        0      0 *:1                     *:*
Active UNIX domain sockets (including servers)
Proto RefCnt Flags       Type       State         I-Node Path . . .
. . .
```

Figure 3.4 Status of the connections as shown by the netstat command

In Floyd's paper [2], it has been concluded that in multiple congested gateways situation, only the bottleneck gateway is the dominant gateway. This conclusion did not contradict with the above result because in this case, the bottleneck gateway was the Cisco 2509 and not the Cisco 4500M. This is because the two clients were not going through a common most congested bottleneck gateway. The basic condition for the fairness problem being able to compare with each other was not satisfied. However, it really reflects one point, topological consideration in real world analysis is very important. The situation may be very complex but finding the real bottleneck gateway before analysis is of prime concern for the fairness problem.

3.3.1 Sample of tcpdump output

A sample tcpdump output is showed in figure 3.5. Measurements are taken from the tcpdump output files. They were used to calculate the throughput at congestion phase and generate plot of sequence number against time.

```
21:56:38.397390 pc.hko.gov.hk.1028 > fairman.ftp: S 975596:975596(0) win 8192 <mss 1460> (DF)
21:56:38.397390 fairman.ftp > pc.hko.gov.hk.1028: S 3142063733:3142063733(0) ack 975597 win 31744 <mss 1460>
21:56:38.597390 pc.hko.gov.hk.1028 > fairman.ftp: . ack 1 win 8760 (DF)
```

```
21:56:38.907390 fairman.ftp > pc.hko.gov.hk.1028: P 1:99(98) ack 1 win 31744 (DF) [tos 0x10]
21:56:39.237390 pc.hko.gov.hk.1028 > fairman.ftp: P 1:17(16) ack 99 win 8662 (DF)
21:56:39.237390 fairman.ftp > pc.hko.gov.hk.1028: P 99:167(68) ack 17 win 31744 (DF) [tos 0x10]
21:56:39.487390 pc.hko.gov.hk.1028 > fairman.ftp: P 17:37(20) ack 167 win 8594 (DF)
21:56:39.487390 fairman.ftp > pc.hko.gov.hk.1028: P 167:214(47) ack 37 win 31744 (DF) [tos 0x10]
21:56:39.587390 fairman.ftp > pc.hko.gov.hk.1028: P 214:369(155) ack 37 win 31744 (DF) [tos 0x10]
21:56:39.977390 pc.hko.gov.hk.1028 > fairman.ftp: . ack 369 win 8392 (DF)
21:56:40.027390 pc.hko.gov.hk.1028 > fairman.ftp: P 37:42(5) ack 369 win 8392 (DF)
21:56:40.027390 fairman.ftp > pc.hko.gov.hk.1028: P 369:400(31) ack 42 win 31744 (DF) [tos 0x10]
21:56:40.207390 pc.hko.gov.hk.1028 > fairman.ftp: P 42:48(6) ack 400 win 8361 (DF)
21:56:40.207390 fairman.ftp > pc.hko.gov.hk.1028: P 400:419(19) ack 48 win 31744 (DF) [tos 0x10]
21:56:40.537390 pc.hko.gov.hk.1028 > fairman.ftp: P 48:72(24) ack 419 win 8342 (DF)
21:56:40.537390 fairman.ftp > pc.hko.gov.hk.1028: P 419:449(30) ack 72 win 31744 (DF) [tos 0x10]
21:56:40.687390 pc.hko.gov.hk.1028 > fairman.ftp: P 72:78(6) ack 449 win 8312 (DF)
21:56:40.697390 fairman.ftp-data > pc.hko.gov.hk.1029: S 2251907932:2251907932(0) win 512 <mss 1460> [tos 0x8]
21:56:40.707390 fairman.ftp > pc.hko.gov.hk.1028: . ack 78 win 31744 [tos 0x10]
21:56:40.957390 pc.hko.gov.hk.1029 > fairman.ftp-data: S 978139:978139(0) ack 2251907933 win 8760 <mss 1460> (DF)
21:56:40.957390 fairman.ftp-data > pc.hko.gov.hk.1029: . ack 1 win 31744 [tos 0x8]
21:56:40.957390 fairman.ftp > pc.hko.gov.hk.1028: P 449:502(53) ack 78 win 31744 (DF) [tos 0x10]
21:56:40.957390 fairman.ftp-data > pc.hko.gov.hk.1029: P 1:367(366) ack 1 win 31744 (DF) [tos 0x8]
21:56:40.957390 fairman.ftp-data > pc.hko.gov.hk.1029: F 367:367(0) ack 1 win 31744 [tos 0x8]
21:56:41.057390 fairman.ftp > pc.hko.gov.hk.1028: P 502:526(24) ack 78 win 31744 (DF) [tos 0x10]
21:56:41.357390 pc.hko.gov.hk.1028 > fairman.ftp: . ack 502 win 8259 (DF)
21:56:42.057390 fairman.ftp > pc.hko.gov.hk.1028: P 502:526(24) ack 78 win 31744 (DF) [tos 0x10]
21:56:42.317390 pc.hko.gov.hk.1029 > fairman.ftp-data: . ack 368 win 8394 (DF)
21:56:42.467390 pc.hko.gov.hk.1029 > fairman.ftp-data: F 1:1(0) ack 368 win 8394 (DF)
21:56:42.467390 fairman.ftp-data > pc.hko.gov.hk.1029: . ack 2 win 31744 [tos 0x8]
21:56:42.467390 pc.hko.gov.hk.1028 > fairman.ftp: . ack 526 win 8235 (DF)
21:56:55.707390 pc.hko.gov.hk.1028 > fairman.ftp: P 78:87(9) ack 526 win 8235 (DF)
21:56:55.717390 fairman.ftp > pc.hko.gov.hk.1028: P 526:555(29) ack 87 win 31744 (DF) [tos 0x10]
21:56:55.867390 pc.hko.gov.hk.1028 > fairman.ftp: P 87:92(5) ack 555 win 8206 (DF)
22:07:06.727390 fairman.ftp-data > pc.hko.gov.hk.1032: . 17521:18981(1460) ack 1 win 31744 [tos 0x8]
22:09:06.727390 fairman.ftp-data > pc.hko.gov.hk.1032: . 17521:18981(1460) ack 1 win 31744 [tos 0x8]
```

Figure 3.5 Sample of fraction of tcpdump output

3.3.2 Sample of dump of kernel variables

A sample dump of kernel variables is showed in figure 3.6. They were used to generate plot of congestion window against time, detailed analysis and monitoring.

```
Mar 17 11:07:05 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 15 17 81
Mar 17 11:07:06 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 15 18 104
Mar 17 11:07:06 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 3 67
Mar 17 11:07:06 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 4 30
Mar 17 11:07:06 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 5 16
Mar 17 11:07:06 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 6 15
Mar 17 11:07:06 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 7 15
Mar 17 11:07:06 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 8 15
Mar 17 11:07:06 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 9 15
Mar 17 11:07:06 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 10 15
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 11 15
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 12 15
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 13 20
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 14 22
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 15 22
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 16 17
```

```
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 17 22
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 18 24
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 19 27
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 20 27
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 21 32
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 22 19
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 23 31
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 24 31
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 25 31
Mar 17 11:07:07 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 26 31
Mar 17 11:07:08 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 27 52
Mar 17 11:07:08 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 28 43
Mar 17 11:07:08 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 29 44
Mar 17 11:07:08 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 30 51
Mar 17 11:07:08 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 31 40
Mar 17 11:07:08 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 32 40
Mar 17 11:07:08 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 33 30
Mar 17 11:07:08 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 34 25
Mar 17 11:07:08 fairman kernel: CONG phase: daddr mss ssthresh cwnd rtt = 17410240 512 1 35 48
```

Figure 3.6 Sample of fraction of dump of kernel variables

3.3.3 Overall throughput comparison

The overall throughput comparison is for convenience only because results reported by the ftp clients can be directly used. By means of overall throughput, we use the throughput as reported by the ftp client for the session. This is not a fair comparison because of the time lag on starting the different sessions, different completion time etc. However, this gives a quick indication of how the results compared. Table 3.1 on the next page summarizes the results for different cases:

Cases	Throughput (amps1, KB/s)	Throughput (amps2, KB/s)	Ratio (amps1/amps2)	Conditions
(a)	800.0	810.0	0.99	*

(b)	519.3	477.4	1.09	∇
(c)	182.7	135.8	1.35	Δ
(d)	189.6	103.1	1.84	ψ
(e)	205.4	120.4	1.71	ψ

Table 3.1 Table summarizing the results for case (a) to case (e)

* No special treatment applied to the gateways.

∇ Traffic is added to the Cisco 4500M router, but no congestion found. The actual command string used is "ping –f –s 512 172.21.80.251" at "metis".

Δ The Cisco 4500M router is made congested by using the ping utility at "metis". The actual command string used is "ping –f –s 768 172.21.80.251", also the F50 run the "ping –f –s 1024 192.168.8.250" command.

ψ The F50 is make further congested by running "ping –f –s 1088 192.168.8.250", this also serves to make the F50 introduces more propagation delay to the rtt of "amps2". Host "metis" still runs the command "ping –f –s 768 172.21.80.251".

Note: Korn shell scripts are driven by the cron daemon at F50, which add randomized traffic into the network for elimination of the Traffic Phase Effect (see section 5.1).

From the above table, we can see that the ratio of throughput for "amps1" over "amps2" under case (c) and (d) ranged from 1.35 to 1.84, indicating that it is pretty unfair for "amps2". The nominal values of rtts for "amps1" and "amps2" are 1.0 and

2.7 ms respectively under case (c) and (d). The explanation for “amps2” receives unfair allocation is as follow:

Under normal situation (no congestion), “amps1” and “amps2” should receive the same bandwidth allocation, therefore their throughput ratio should be one. As case (a) and case (b) shown, this is in fact the case. Case (b) suggested that even under a light load traffic situation, the allocation is still fair since the ratio is still very close to one. But for the congestion case, case (c) and (d) told us that it deviated from one. The ratio is amps1 over amps2, so with a ratio greater than one implied that “amps1” receives more bandwidth while the reverse is true for “amps2”. In other words, “amps2” receives unfair bandwidth allocation in this case.

The purpose of case (e) is to test the hypothesis that initial window set up before congestion occurs may have effects on subsequent throughput. The result of case (e) indicates that this is not significant.

It should be noted that actually there are many sets of experimental data, Table 3.1 gives typical values for sake of simplicity. The next page shows selected plots for data from the measurements.

3.3.4 Plot of sequence number (seq#) against time

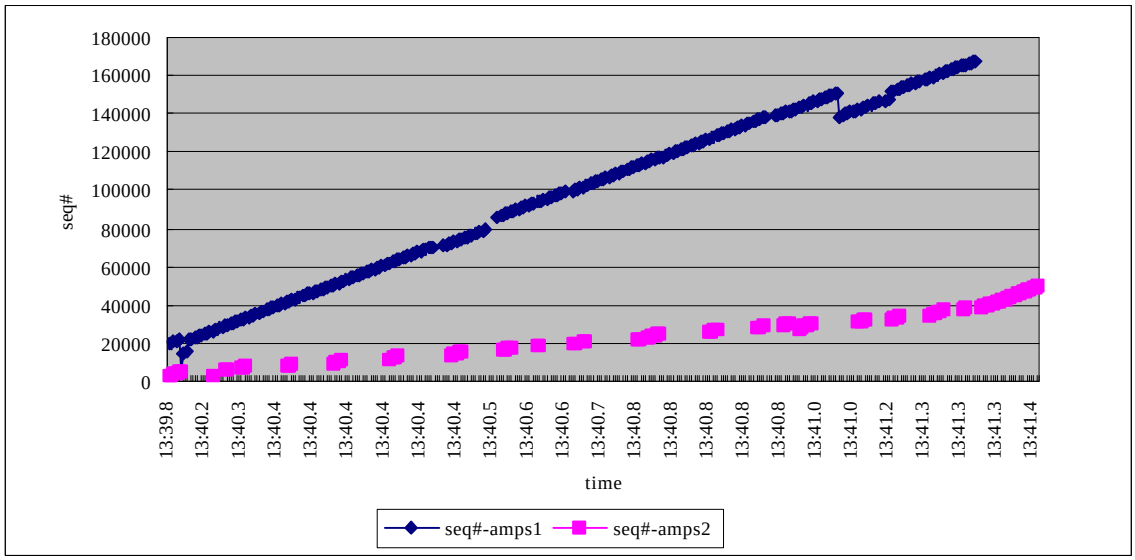


Figure 3.7 Plot of seq# vs t for “amps1” and “amps2” under congestion

From this graph, we can clearly see that the slope for the sequence number against time is greater for “amps1” (dark line) than “amps2” (grey line). Retransmissions can also be identified (e.g. the broken dark line at around 13:41.0 in fig.3.7).

3.3.5 Plot of congestion window (cwnd) against time

There are several interesting points as shown in figure 3.8. First, when congestion occurs, both “amps1” (the dark line) and “amps2” (the grey line) get the same signal for congestion, the only difference is the degree of congestion as seen by them. This is expected because the signal for congestion is packet losses which both experience with the same probability if Traffic Phase Effect is eliminated (see section 5.1). Second, in this plot the rtts for “amps1” and “amps2” are around 1 ms and 2.7 ms respectively. Anyway, with a difference of only 1.7 ms, we can see that the plotted line of “amps2” is always below that of “amps1” and occasionally drops its cwnd to half due to higher

degree of congestion as seen by “amps2”. Also the chance for “amps2” entering into the slow start phase, having a congestion window of one, is much larger than that for “amps1”. Third, “amps1”’s cwnd always grows faster than “amps2” as reflected by the slope of the plot. Additionally, “amps1” always has a larger maximum cwnd magnitude as compared to “amps2”.

The conclusion is that, due to the TCP window increase algorithm for congestion avoidance; there is preference for “amps1”, the shorter rtt client, over “amps2”, the longer rtt client. Hence, the fairness problem is identified.

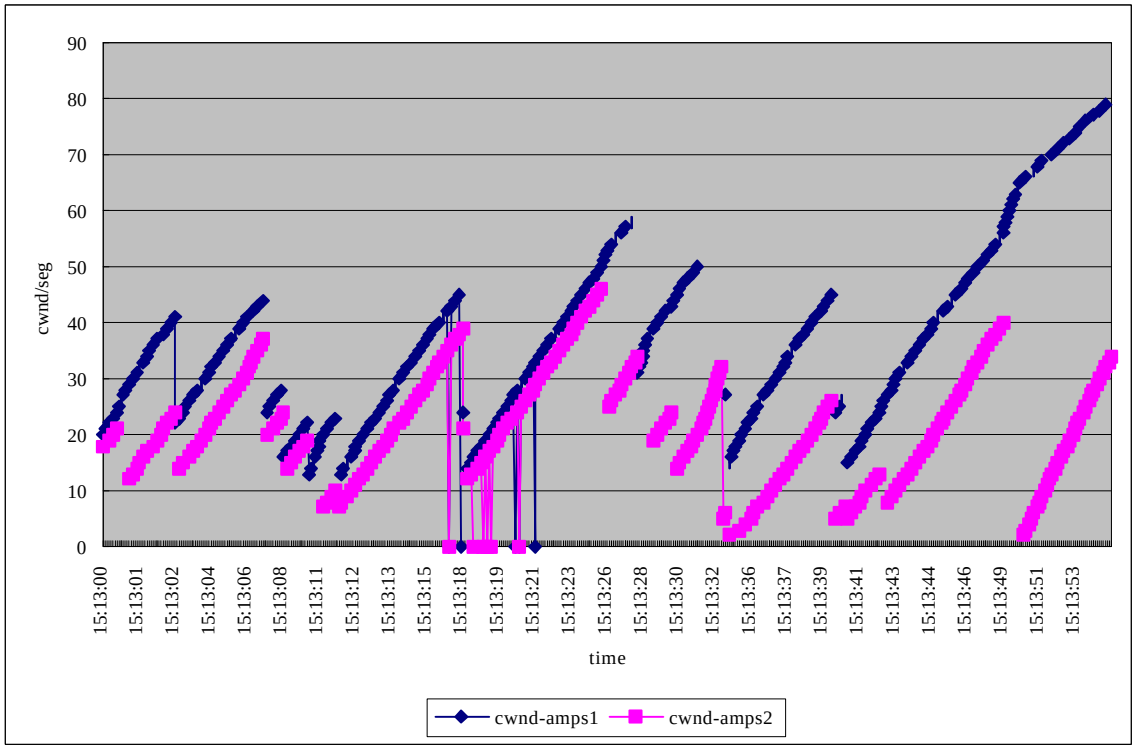


Figure 3.8 Plot of congestion window (cwnd) against time for “amps1” and “amps2” under congestion

3.3.6 Throughput comparison at congestion phase

Cases	Throughput (amps1, KB/s)	Throughput (amps2, KB/s)	Ratio (amps1/amps2)	Conditions
(c)	177.2	126.8	1.40	Δ
(d)	194.0	97.0	2.00	ψ

Table 3.2 Table summarizing the results for case (c) and (d)

The throughput comparison at the congestion phase reflects the actual competition between different clients. They are deduced from the tcpdump output by only considering the period when congestion occurs. Each output is examined on a case by case basis to avoid mis-interpretation. As can be seen from the above table, the actual ratio is higher, indicating in reality it is more unfair for "amps2" in this case. The ratio of overall throughput comparison although not showing the actual competition, is a convenient value for comparison of fairness. In fact for the case of exact equal allocation of bandwidth with clients of same rtts, the ratio should approach one regardless of whether the overall throughput ratio or throughput ratio during congestion is used.

3.3.7 Conclusion

From all the results shown above, the fairness problem is clearly identified. A client with longer rtt will suffer a lower throughput when connected through a bottleneck gateway to a server. Although only a simple topology is being experimented, larger

TCP/IP networks can be forecasted to exhibit the same problem under suitable scenarios. For larger TCP/IP networks, careful examination of the topology under consideration is emphasized.

Chapter 4 **Finding Improvements**

From chapter 3, the evidence for negative throughput bias against long rtt is presented. The next step is to find a solution for this fairness problem. As mentioned in section 2.3, the linear rate and constant rate window increase policies are possible solutions. These two algorithms change the rate of growth of the congestion window in order to gain more bandwidth during congestion for restoring fairness. The sections below describe its implementation and the experimental result.

4.1 The Modified Algorithm

As we have said before, the bias against long rtt stems from the "congestion avoidance" algorithm used by TCP. Therefore the most obvious solution is to modify this algorithm. Two proposals have been discussed in chapter 2 and simulated by some researchers, namely the "linear rate" and "constant rate" window increase policies. These algorithms have now been implemented in a Linux 2.0.27 kernel in the current project for experimentation.

In almost all existing implementation of TCP, the window increment algorithm executes the following command each time a new acknowledgement is received:

cwnd = cwnd + 1 / cwnd

The algorithm increases cwnd by one segment for each rtt. To modify the algorithm, the numerator of the fractional part of the above equation is changed to c*rtt for the "linear rate", while for the "constant rate" case, it is changed to c*rtt*rtt. The constant c is experimented with several different values to see its effect on the performance.

4.2 Implementation

In Linux, the kernel maintains the cwnd in unit of segments rather than in bytes as contrast to most implementations. In order to make the increment in terms of segment, a counter is maintained for this purpose. Inside the program, conditional check on this counter decides whether there should be increases or not. Part of the original coding is shown below for reference:

```
/*
 *      We don't want too many packets out there.
 */

if (sk->ip_xmit_timeout == TIME_WRITE &&
        sk->cong_window < 2048 && after(ack, sk->rcv_ack_seq))
{

        /*
         * This is Jacobson's slow start and congestion avoidance.
         * SIGCOMM '88, p. 328.  Because we keep cong_window in integral
         * mss's, we can't do cwnd += 1 / cwnd.  Instead, maintain a
         * counter and increment it once every cwnd times.  It's possible
         * that this should be done only if sk->retransmits == 0.  I'm
         * interpreting "new data is acked" as including data that has
         * been retransmitted but is just now being acked.
         */
        if (sk->cong_window <= sk->ssthresh)
```

```
                /*
                 *      In "safe" area, increase
                 */
                sk->cong_window++;
        else
        {
                /*
                 *      In dangerous area, increase slowly.  In theory this is
                 *      sk->cong_window += 1 / sk->cong_window
                 */
                if (sk->cong_count >= sk->cong_window)
                {
                        sk->cong_window++;
                        sk->cong_count = 0;
                }
                else
                        sk->cong_count++;
        }
}
```

To implement the modified algorithms, we rewrote the final “else” statement as shown below:

4.2.1 Linear rate policy

The “else sk->cong_count++;” is changed to

“else sk->cong_count += (sk->rtt/30);”

The constant 30 here is referred as c’ in this case. Its value will be experimented.

4.2.2 Constant rate policy

The “else sk->cong_count++;” is changed to

“else sk->cong_count += sk->rtt * sk->rtt / 1000);”

The constant 1000 here is also referred as c' in this case. Its value will also be experimented.

After making these modifications, the kernel is then recompiled and installed for experimentation.

As one can see from the above codes, this is basically equivalent to modifying the numerator in the equation cwnd=cwnd+1/cwnd, except that the unit of increment is maintained in segments rather than bytes. The congestion counter is updated upon acknowledgement and if the condition of congestion counter is greater than or equal to the congestion window is met, then the congestion window and congestion counter will be updated and reset respectively.

4.3 Result

4.3.1 Overall throughput comparison

Results obtained after modifying the kernel are summarized in the following tables:

Constant c' used	Throughput (amps1, KB/s)	Throughput (amps2, KB/s)	Ratio (amps1/amps2)	Conditions
10	190.5	102.4	1.86	ψ
15	177.8	104.6	1.70	ψ
20	184.1	126.5	1.46	ψ

30	167.1	132.9	1.26	ψ
30	156.7	132.9	1.18	Δ

Table 4.1 Table summarizing the results for “linear rate” policy

Constant C’ used	Throughput (amps1, KB/s)	Throughput (amps2, KB/s)	Ratio (amps1/amps2)	Conditions
3000	162.0	119.2	1.36	ψ
2000	161.6	121.7	1.33	ψ
1500	153.7	130.1	1.18	ψ
1000	171.8	136.4	1.26	ψ
1000	150.1	136.0	1.10	Δ

Table 4.2 Table summarizing the results for “constant rate” policy

Ø,Ä’s conditions are the same as those described in chapter 3.

From Table 4.1 and 4.2, we can see that both the “linear rate” and “constant rate” policies restore the fairness to a certain extent subject to how the constant c’ is selected. As shown in the above tables, the constant c’ is important because its value can affect the fairness.

In fact, the need of selecting a good constant c’ is far apart from ideal. This constant may be different under different situations for optimal performance. At least, c’ should be adaptive according to the situation.

4.3.2 Plot of cwnd against time

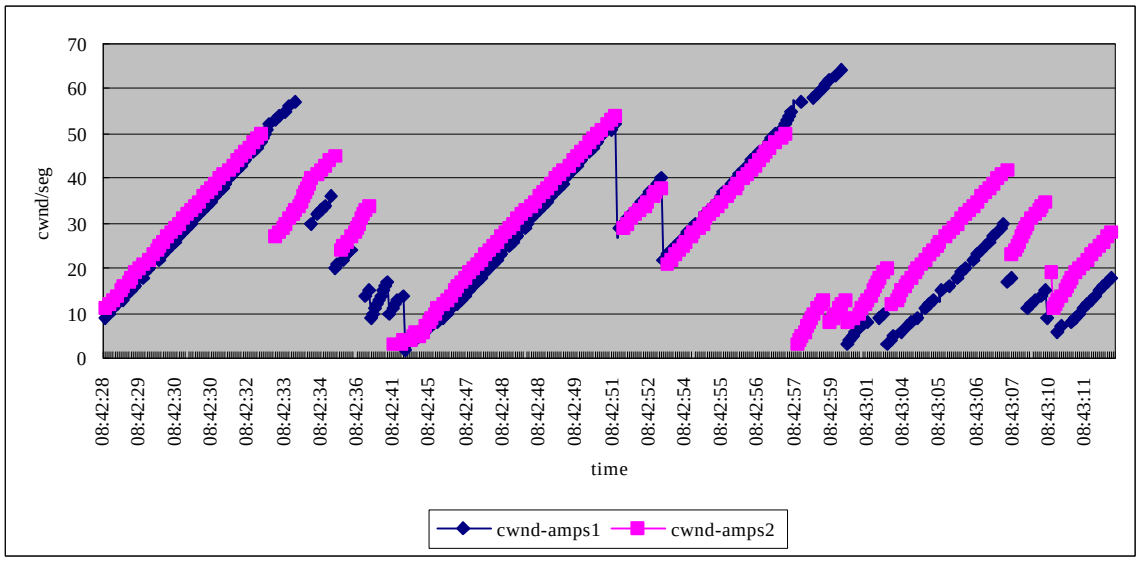


Figure 4.1 Plot of cwnd vs t for the “linear rate” kernel with c’=30 under heavy loading

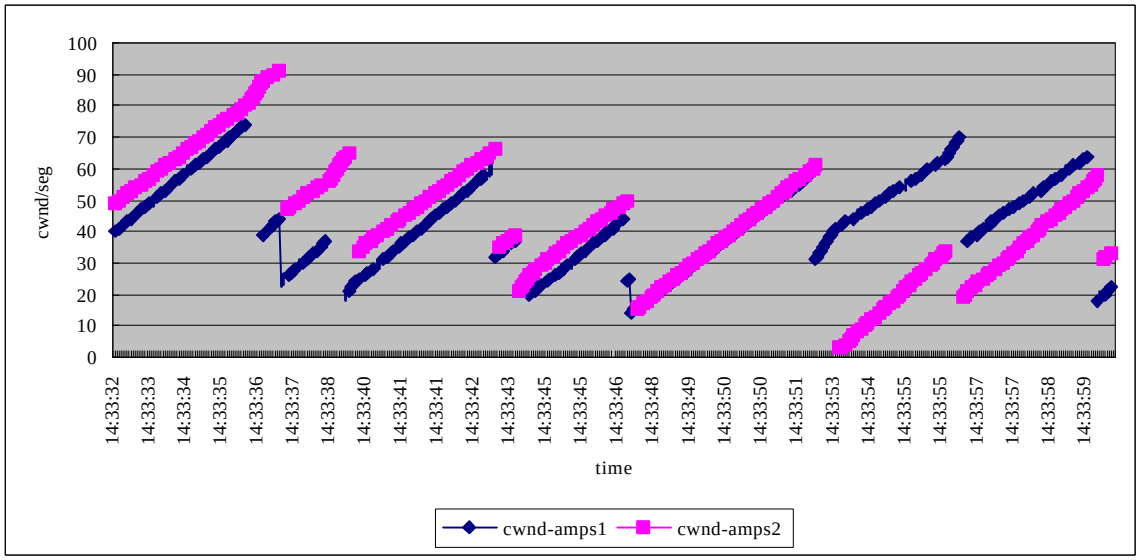


Figure 4.2 Plot of cwnd vs t for the “constant rate” kernel with c’=1500 under heavy loading

In the above two figures, dark line refers to cwnd of “amps1” while grey line referred to cwnd of “amps2”.

In chapter 3, the plot of cwnd vs time (fig. 3.8) shows that the plotted line of “amps2” is always below that of “amps1”. In other words, the growth rate of cwnd for “amps2” is less than that of “amps1” causing the unfairness. In Figure 4.1 and 4.2, this phenomenon is rectified, at least to a certain extent. In some cases, the growth rate of cwnd of “amps2” even runs over those of “amps1”, indicating some degree of fairness being restored. For exact fairness, the slope of the curves should be the same. By examining Figure 4.1 and 4.2, one can observe that the “constant rate” is somewhat more aggressive.

4.3.3 Throughput comparison at congestion phase

Constant c’ used	Throughput (amps1, KB/s)	Throughput (amps2, KB/s)	Ratio (amps1/amps2)	Conditions
1000	176.0	118.8	1.48	ψ
1000	141.4	119.7	1.18	Δ

Table 4.3 Table showing the effective throughput comparison for the “constant rate” policy at c’=1000

Once again, it can be seen that the throughput ratio during congestion are higher than those overall throughput ratio.

4.4 Discussion

From Tables 4.1, 4.2 and case (d), one fact was observed from these data; no matter it is the “linear rate” or “constant rate” method, fairness is restored by injecting more packets into the network which has the effect of hurting the other in order to gain benefit for itself. As one can observe, the throughput for “amps1” was lowered whenever the ratio amps1/amps2 approaches one. This is not desirable because injecting more packets, although in a controlled manner, means inducing more congestion. These algorithms do not exhibit disaster effect because the “slow start” of TCP release the congestion but the cycle repeats again and again. The overall result is that bandwidth is wasted since the connections cannot attain steady state (or equilibrium) for a substantial period. Also the gateway needs to drop packets more frequently and the source needs retransmission more frequently. Nevertheless, the “linear rate” and/or “constant rate” can be viewed as an interim measure for restoring the fairness of throughput. A better algorithm to address the fairness issue is obviously urgently needed especially for networks of long round trip delay.

Chapter 5 Miscellaneous Issues

This chapter describes the Traffic Phase Effect [1], Bursty Traffic Effect [2], Gateway Effect [2] and the correlation between rtt and cwnd, because they are either directly or indirectly contributing to the fairness problem. The Traffic Phase Effect is not directly related to the fairness issue, but it is important to eliminate this effect during experiment. Otherwise measurements may be contaminated and subsequent interpretation and analysis may therefore lead to wrong conclusion. For sake of completeness, we try our best to describe all related issues.

5.1 Traffic Phase Effect

Traffic Phase Effect originates from the periodic nature of data transfer at the congested state, especially for bulk data transfer. The underlying reason roots from the window flow control protocols which has a periodic cycle equal to the connection's rtt (e.g. a network-bandwidth limited TCP bulk data transfer). Control theory suggests that this periodicity can resonate with deterministic control algorithms in network gateways. The final result is that TCP/IP network with Drop Tail gateway (basically a FIFO gateway) when coupled with strongly periodic traffic would end up with systematic discrimination against some connections. However, this discrimination can be eliminated with the addition of appropriate random traffic to the network. In particular, gateways that drop packets randomly on overflow (Random Drop Gateway)

are sufficient to overcome this effect. Details on this effect are described in Floyd's paper [1]. The important point here is this, phase effects can result in performance biases in networks and in network simulations. Figure 5.1 below illustrates the Traffic Phase Effect for a two-node network as shown in Figure 5.2.

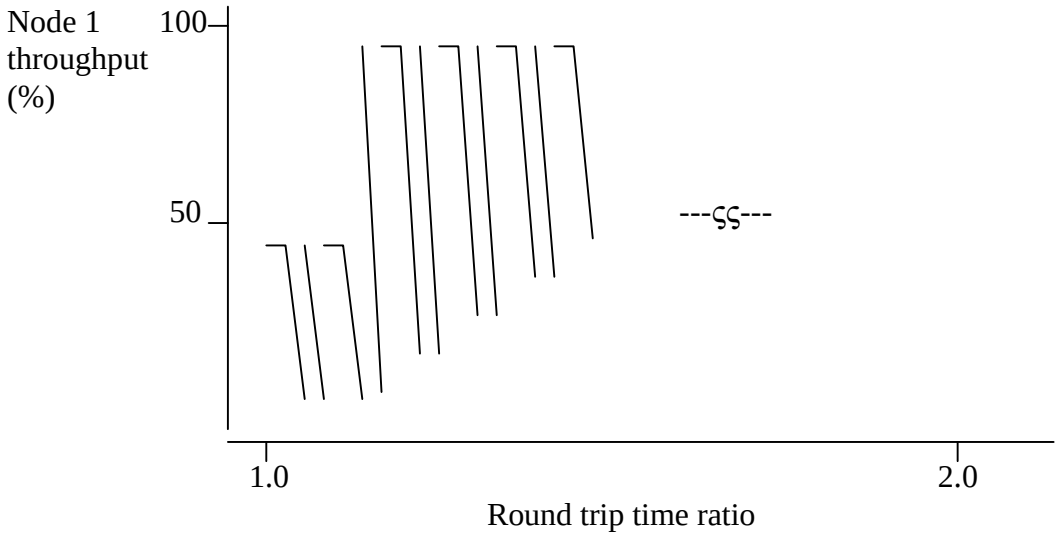


Figure 5.1 Node 1's throughput as a function of node 2's rtt

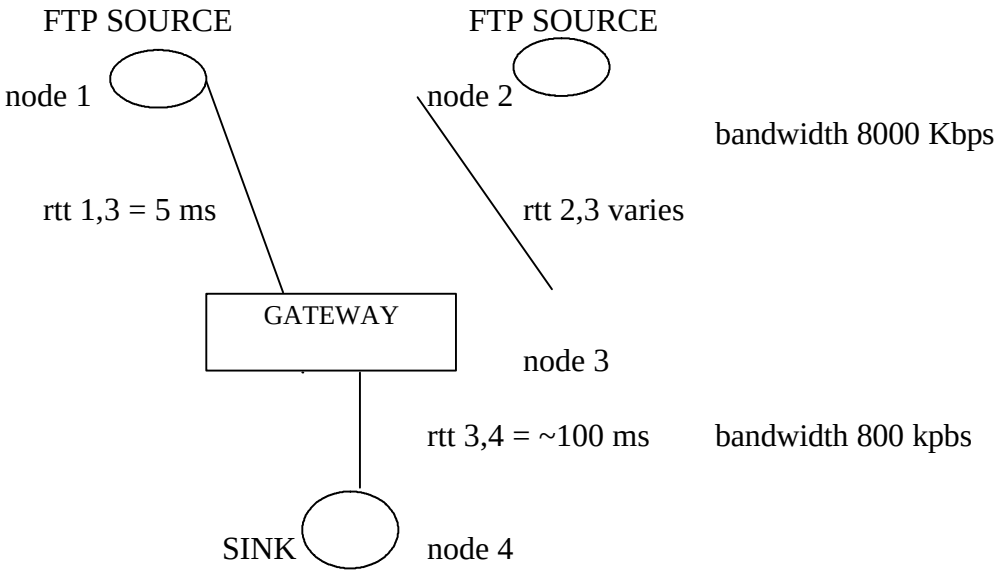


Figure 5.2 The simulation network

From figure 5.1, we can see that the throughput of node 1 varies as a function of the rtt ratio of node 1 and node 2. The difference is quite significance, ranging from over 80% to less than 20% throughput. Obviously this effect can not be ignored during measurements and simulations. In our experiment, this effect is minimized by introducing random traffic into the network in order to destroy the periodicity of the ftp source.

5.2 Bursty Traffic Effect

TCP/IP networks with either Drop Tail or Random Drop gateways have a bias against bursty traffic. By bursty traffic, it means traffic from connections where the current window is small when compared to the bandwidth-delay product, or connections where the amount of data generated in one rtt is small when compared to the bandwidth-delay product. This may often result from loose cluster of packets transmitted when the gateway's queue is likely to overflow. This bias occurs because the contents of the gateway queue when the queue overflows are not necessarily representative of the average traffic through the queue. Bias against bursty traffic becomes more severe with Drop Tail and Random Drop gateways. Simulation showed that the throughput for bursty traffic can be improved with gateways such as Random Early Detection (RED) gateways, which detect incipient congestion early. With a RED gateway, a node that transmits packets in a cluster does not have a disproportionate probability of having a dropped packet. Bursty Traffic Effect is found to be in-significant in this project's measurements.

5.3 Gateway Effect

During congestion, gateway influences the fairness problem because its queuing discipline determines which packet to drop. This has effect on retransmissions used by TCP. Normal or default configuration of gateways used the FIFO discipline, commonly referred to as Drop Tail (DT) gateway. Other possible gateway's configuration of discipline include the Random Drop (RD) Gateway and the Random Early Detection (RED) Gateway. There are reported benefits of RD gateway over DT gateway. The benefits include fairness to late-starting connections and slightly improved throughput for connections with longer rtt. However, there are also reported shortcomings of the RD gateway. They include the preferential treatment for connections with shorter rtt, a higher throughput for connections with larger packet sizes, and a failure to limit the throughput for connections with aggressive TCP implementations. Anyway, these shortcomings are shared by the DT gateways also.

Theoretical analysis shows that the rtt bias in TCP/IP networks results from the TCP window increase algorithm and not from the gateway dropping policy. However, the traffic phase effect is highly correlated with the gateway policy. The result in Table 5.1 demonstrates that gateway effect is not significant for the fairness problem provided that the traffic phase effect is eliminated:

Constant c' used	Throughput (amps1, KB/s)	Throughput (amps2, KB/s)	Ratio (amps1/amps2)	Conditions
1000	171.8	136.4	1.26	ψ FIFO gateway
1000	185.1	135.1	1.37	ψ RED gateway
Normal kernel	177.2	99.96	1.77	ψ RED gateway

Table 5.1 Table showing the overall throughput for different gateways

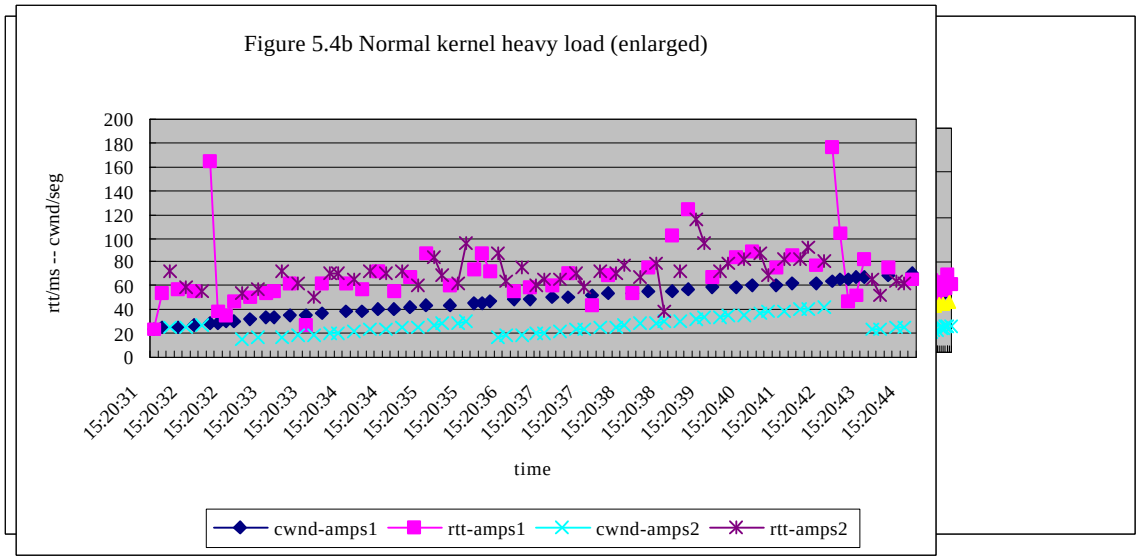
ψ : the same condition as described in chapter 3.

As can be seen from the above table, the bias against longer rtt still exists for a normal TCP kernel and the "constant rate" policy restores the fairness to a certain extent. The RED gateway configuration used is the default of Cisco's implementation. The difference between RED and FIFO gateway is not significant. The gateway effect therefore should not affect the overall result and conclusion in our experiment.

5.4 Positive correlation between Round Trip Times and Congestion Window

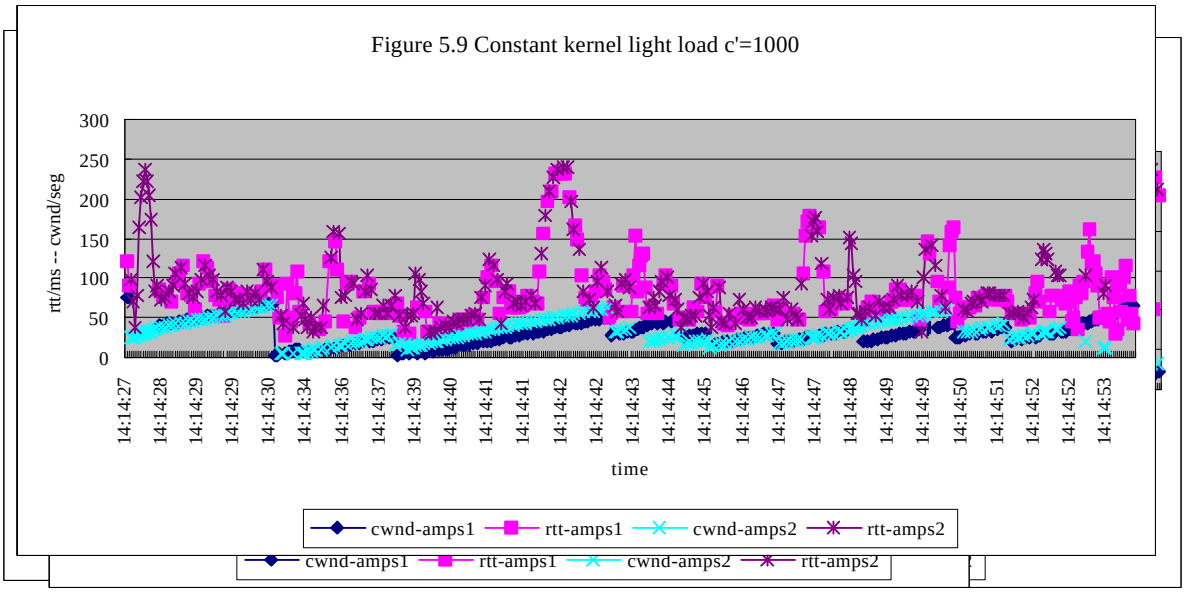
During the experiment, it has been observed that the rtts, especially during congestion, vary greatly but following the pattern of the growth of the congestion window (for example, please see Figure 5.5a and 5.5b). From a theoretical viewpoint, this is expected since traffic increases as the congestion window increases. However, this phenomenon has not been mentioned in all the related papers. Basically, it is suspected that because the rtt is a parameter required for simulation, it is hard-wired in the simulator and not treated as a variable. If this is the case, then all the simulation results should subject to certain distortion because the rtt is treated as an invariant. The possible outcomes are discussed at the next section. The following graphs illustrates

Figure 5.4b Normal kernel heavy load (enlarged)



the variation of rtts during congestion:

Figure 5.9 Constant kernel light load c'=1000



5.5 Possible Outcomes

With the correlated variation of rtts with congestion window, one can infer that both the “linear rate” and “constant rate” methods will proportionally inject more packets into the network in order to hunt for more bandwidth. When congested, rtt shots up. Since the rate is proportional to c*rtt*rtt for “constant rate”, the effect will be more severe for that case. The end result is that, when the network can not bear any more, slow start will be the only fallback mechanism. Imaging for a global TCP/IP network, a bottleneck gateway shared many connections in congestion with say over ten long rtt connections. The algorithm will make those long connections together with others run into slow start due to too much traffic. Even worse, all the connections may oscillate in this case. Certainly, this is not an acceptable scenario.

5.6 Proposal

To cope with the difficulty for restoring fairness while not over congesting the network at the same time, a better congestion avoidance algorithm is needed. In short, the proposed algorithm should obey the following criteria:

- Rate of injection should not be directly related to rtt (the fairness issue)
- Convergence to steady state quickly
- Try to maintain at steady state and avoid oscillation

- Use rtt as a signal for correction, because this is the only available signal in the current TCP/IP design
- Any parameter similar to c or c’ (as those used in section 4.2.1 and 4.2.2), if used, should be adaptive.
- Should be able to incorporate into current TCP implementations without significant modification.

Chapter 6 A new algorithm and the result

Taking into account the finding discussed in section 5.4 and the proposal discussed in section 5.6, we try three new algorithms and the results presented in section 6.3. The first two algorithms are just an extension of the previous “linear rate” and “constant rate” policies with c’ being adaptive. The third algorithm mainly reflects the points considered in the proposal discussed in section 5.6, using rtts during congestion as a predictor to avoid further congestion. Surprisingly, this alleviates congestion a lot. However, for the third algorithm to work, it has to be applied in a global scale meaning that every implementation has to obey the same rule.

6.1 The new algorithms

One common feature of these three new algorithms is that the constant c’ is adaptive. The present method scale c’ linearly. As rtt increases, c’ increases in a step of certain value (depending on whether it is the constant rate or linear rate). The reverse is c’ decreases in a step of the same value as recent rtt decreases. The three new algorithms tested are briefly described below:

A. Constant rate with c’ adaptive; that is c’ increases as rtt increases and vice versa,

B. Linear rate with c’ adaptive; that is c’ increases as rtt increases and vice versa,

C. This algorithm monitors the variation of rtt during congestion, making linear increments with c' adaptive (c' increases as rtt increases and vice versa) only when the trend of rtt is dropping.

6.2 Implementation

To implement the first new algorithm, two static integer variables are declared inside the kernel, namely, ortt initialized to 1 and c initialized to 1500. It is as follows:

The "else sk->cong_count++;" is changed to

```
"else {
        if (sk->rtt > ortt) {
                if (c > 2300) c=2300;  /* ceiling value */
                else c += 150;          /* the fixed step under investigation */
                sk->count += (sk->rtt*sk->rtt/c);       /* constant rate */
        }
        else {
                if (c < 700) c=700      /* floor value */
                else c-= 150;           /* the same fixed step used */
                sk->count += (sk->rtt*sk->rtt/c);       /* constant rate again */
        }
        ortt = sk->rtt;                 /* record the recent rtt for next
comparison */
```

}”

To implement the second new algorithm, two static integer variables are declared inside the kernel, namely, ortt initialized to 1 and c initialized to 40. It is as follows:

The “else sk->cong_count++;” is changed to

“else {

```
if (sk->rtt > ortt) {
        if (c > 55) c=55;          /* ceiling value */
        else c += 2;               /* the fixed step under investigation */
        sk->count += (sk->rtt/c);  /* linear rate */
}
else {
        if (c < 25) c=25;          /* floor value */
        else c -= 2;               /* the same fixed step used */
        sk->count += (sk->rtt/c);  /* linear rate again */
}
ortt = sk->rtt;                    /* record the recent rt for next
```

comparison */

}”

To implement the third new algorithm, three static integer variables are declared inside the kernel, namely, c initialized to 40, ortt1 and ortt2 both initialized to 1.

The “else sk->cong_count++;” is changed to

```
“else {
        if (sk->rtt < ortt2 && sk->rtt < ortt1) {  /* trace the short history */
                if (c < 25) c=25;          /* floor value */
                else c -= 2;               /* the fixed step under investigation */
                sk->count += (sk->rtt/c);       /* linear rate used */
        }
        else {
                if (c > 55) c=55;          /* ceiling value */
                else c += 2;               /* the same fixed step used */
        }               /* note: no linear rate increment any more here */
        ortt2 = ortt1;
        ortt1 = sk->rtt;          /* record those recent rtts for decision making */
}”
```

The essential difference of algorithm C with A and B is that it stops the growth of cwnd if any one of the previous two recorded rtts during congestion is found to be greater than the current recorded rtt. In other words, cwnd increases only when the current recorded rtt during congestion is found to be smaller than the previous two recorded rtts, indicating at least a partial release of congestion. Also the linear policy is used for fairness to reduce level of congestion, the constant c used is adaptive to let itself tune for the proper environment. The present method is to let c increases in a fixed step as rtt increases and vice versa as rtt decreases. Of course, this type of linear adjustment may be too simple and need further improvement. However, the idea is of prime importance as can be shown by the result in section 6.3. Although this algorithm

is not ideal in the current stage, it points out the weak point of current TCP implementations and suggests a new direction of thinking for congestion avoidance. It also sheds light on future TCP implementations for congestion avoidance and fairness. The current implementations for these three algorithms use a variable c similar to c', but the values are bounded to a ceiling value and a floor value. This is to eliminate the case for c to increase or decrease without bound, or changed to some undesirable values such as zero. Effects and implications of doing so need further investigation which is out of the scope of the current project.

6.3 Result

6.3.1 Overall throughput comparison

Algorithm	Throughput (amps1, KB/s)	Throughput (amps2, KB/s)	Ratio (amps1/amps2)	Conditions
A	188.5	135.2	1.39	ψ
B	179.0	131.7	1.36	ψ
C	211.4	130.5	1.62	ψ

Table 6.1 Table summarizing the results for the three new algorithms

Description	Throughput	Throughput	Ratio	Conditions

	(amps1, KB/s)	(amps2, KB/s)	(amps1/amps2)	
Case (d) Original TCP Kernel	189.6	103.1	1.84	ψ
Constant kernel c' = 1500	153.7	130.1	1.18	ψ
Linear kernel C' = 30	167.1	132.9	1.26	ψ

Table 6.2 Table showing previous results (chapter 3 & 4) for easy comparison with Table 6.1.

ψ's conditions are the same as those described in chapter 3.

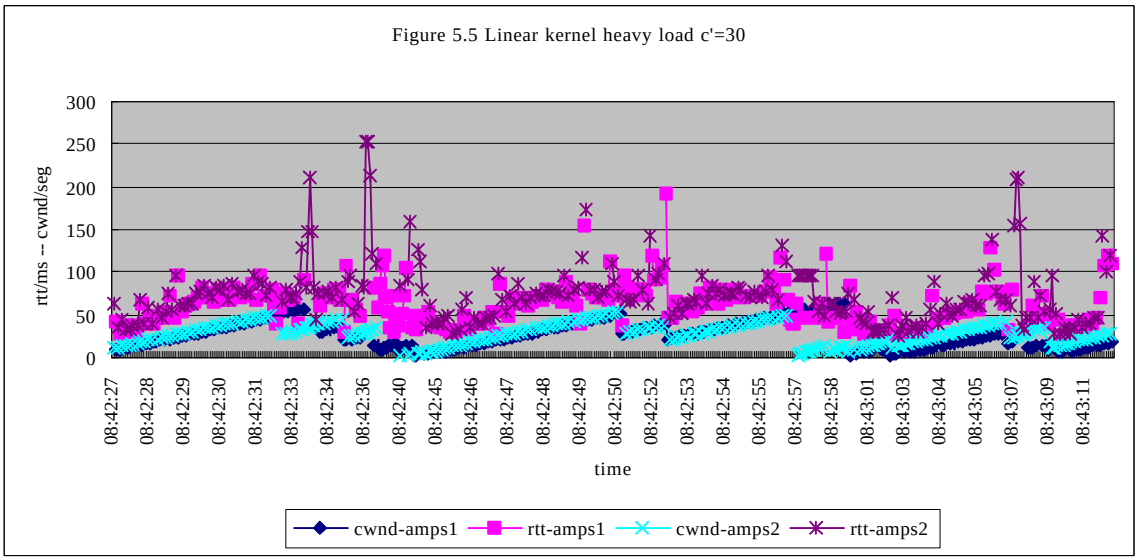
Comparing Tables 6.1 and 6.2, several points are identified. First of all, except for the case (d), which is the original TCP kernel without any modification; the throughput for host "amps2" under all cases lies around 131 KB/s as compared to 103.1 KB/s of the original kernel. Therefore there are improvement for throughput in all cases. As regards to fairness, all are below 1.84 of case (d), so there are improvement of fairness with different extent. The lowest one is the "constant kernel" case with c'=1500 in the above figure, indicating greatest fairness in this set of experiment. However, the fairness is by decreasing the throughput of host "amps1", even below that of the original kernel's case. The same is true for the "linear kernel" case. Hurting others in order to gain the benefit of itself, this sort of fairness is not acceptable at all. Looking at the algorithm corresponding to cases A, B and C, they all have throughput for host

"amps1" higher than the original kernel's case while maintaining the throughput of host "amps2" in approximately the same level with those of the "linear kernel" and "constant kernel" cases. Is it a much better scenario? Is it a better method for the whole that participates in the game? Algorithm C now seems to be the best among those experimented. Of course, there are rooms for further improvement. But first of all, we need to find an explanation for such a change. A possible answer will be given in the next section.

6.3.2 Plot of cwnd and rtt against time

The variation of rtts and cwnd against time for algorithm C is shown in Figure 6.1. Figure 5.5 is reproduced here also for easy comparison. The level of congestion for this algorithm is low when compared with the original TCP kernel and those of others experimented. This is reflected by the number of times the routine for congestion avoidance is called as recorded by the kernel's variables dump files (typically a few hundreds as compared to a few thousands). A point needs to be clearly explained here is that, the plot of figure 6.3 is a plot of the whole transfer while figure 5.5 only shows a fraction. As usual, dark line refers to those data of host "amps1" and grey line (or dots) for host "amps2". By looking at the two figures, we can see that the average of rtt for algorithm C lies somewhere between 25 and 75; while those for the "linear kernel" case lies somewhere between 50 and 100. This again shows that the level of congestion is lower for algorithm C if the rtts can be used as a predictor for the level of congestion. Also, the growth of cwnd of host "amps2" occasionally runs over that of host "amps1" for algorithm C, while they are approximately the same for the "linear

kernel" case. The pattern of plot of rtt shifted, suggesting the resultant dynamics of



TCP changes as a result.

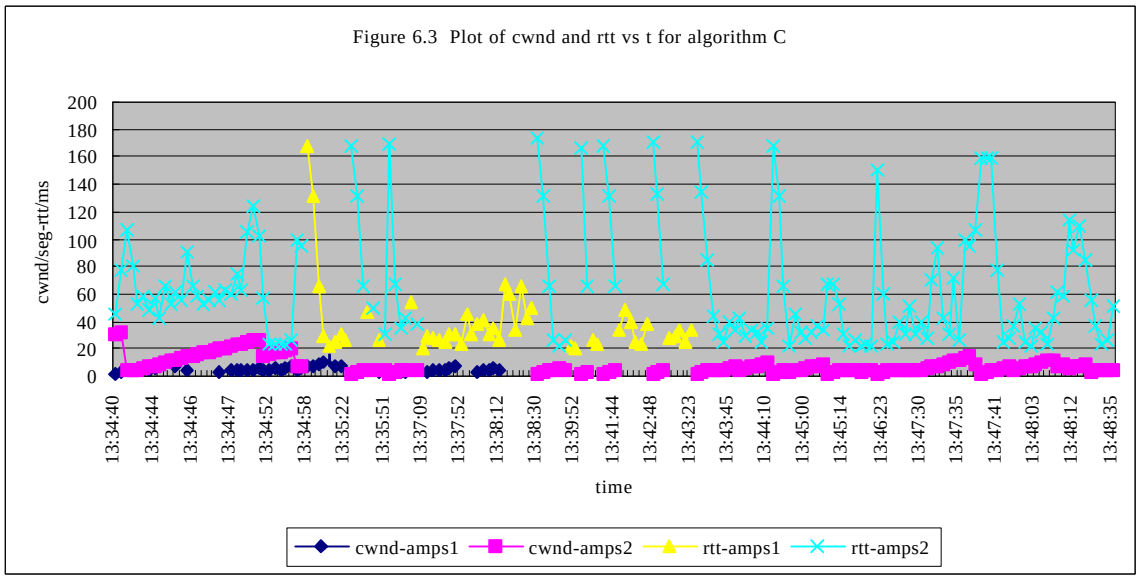


Figure 6.1 Plot of cwnd and rtt vs t for algorithm C

Also, a remarkable feature is that host "amps1" runs into the congestion avoidance phase for algorithm C fewer than the "linear kernel" case.

A plausible explanation for the above features is as below:

Algorithm C mainly reduces the level of congestion by monitoring the trend of recent rtts recorded. If it is in the rising trend indicating possible further congestion, then the growth of cwnd is stopped, otherwise it is increased by the "linear rate" policy to account for fairness. As compared to the original kernel, which always increased in a linear fashion upon acknowledged, algorithm C is more superior and the throughput is actually improved. The guessed reason behind is that algorithm C used an orderly dispatching mechanism with rtt as a predictor rather than increasing the window blindly. It is similar to an analogy of many people trying to go through a narrow gate. If all try to rush to the gate and get through, the gate is stuffed and only very few strong people can get through, others even get hurt. If there is a policeman to maintain the order and people queue up properly, the flow is smoothed and people can get through more easily. Monitoring of rtts here is roughly equivalent to the policeman, allowing people to go batch by batch rather than rushing. Moreover, one more possible effect is that the reduction in level of congestion allows acknowledgements coming back more easily rather than being dropped by the bottleneck gateway. This in turns help to trigger more packets to send and maintain the pipeline moving. Practically, as judged by the result, this algorithm works nicely. The important finding here is that; the original TCP congestion avoidance algorithm, without any regulatory mechanism for release of congestion exactly during congestion, is considered as an unsatisfactory method for congestion control (not to state, the fairness issue also).

Chapter 7 Conclusion and Future Work

7.1 Conclusion

The fairness problem has been observed and identified experimentally. The original TCP implementation cannot rectify the problem, also it is found that the original implementation by itself was far from ideal. It is unable to utilize the maximum possible bandwidth available at the time during congestion. Furthermore it has been found that the round trip times during congestion bears a positive correlation with the growth of the congestion window. The “linear rate” policy and “constant rate” policy as proposed by other researchers have been implemented and tested. They are found to be able to rectify the fairness problem to a certain extent but in expense of the other competing clients’ bandwidth. An algorithm has been devised, implemented and tested, taking into account the correlation of round trip times with congestion window. The result is found to be very encouraging. This algorithm is able to dig additional bandwidth as compared to the original TCP congestion avoidance algorithm as well as restoring fairness to a certain extent. This provides additional insight into the TCP dynamics.

The following table summarizes this work and all previous as a short review:

Description	Previous work	Work in this thesis
Linear rate policy	By simulation	Implemented and tested in a testbed network
Constant rate policy	By simulation	Implemented and tested in a testbed network
Linear rate policy with constant c adaptive	nil	Implemented and tested in a testbed network
Constant rate policy with constant c adaptive	nil	Implemented and tested in a testbed network
A totally new algorithm to address the fairness issue	nil	Implemented and tested in a testbed network

7.2 Future Work

There are plenty of rooms for further developments and researches. They mainly lie in the following areas:

- Improvement of the congestion avoidance's algorithm in order to optimize the bandwidth allocation and fairness.
- Study the effect of variable c and determine its proper range for operation.

- Investigating in a larger scale how the TCP dynamics varies with regard to this problem and others.
- Investigation of the behaviour of TCP's fairness issue under a multi-protocols environment.
- Gateway's behaviour with different queuing strategies with regard to the changes in TCP's behavior.
- Include different TCP implementations and investigates the resultant dynamics.
- Investigation of the behaviour of different application protocols using TCP as transport, for example http and telnet etc.
- Studying the multiple two-way connections and its corresponding dynamics.

Bibliography

[1] Sally Floyd and Van Jacobson, "Traffic Phase Effects in Packet-Switched Gateways", Computer Communications Review, V.21 N.2, April 1991, p26-42.

[2] Sally Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks, Part I: One-way Traffic", Computer Communications Review, V.21 N.5, October 1991, p30-47.

[3] Douglas E. Comer. Internetworking with TCP/IP, volume I: Principles, Protocols, and Architecture. Second Edition. Prentice-Hall International, Inc. 1991.

[4] W. Richard Stevens. TCP/IP Illustrated, Volume I, The Protocols. Addison-Wesley Publishing Company, 1994.

[5] J. Postel, "Transmission Control Protocol. Request for Comments 793". http://www.nexor.com/public/rfc/rfc.html

[6] V. Jacobson, "Congestion avoidance and control". In Proceedings of the ACM SIGCOMM '88, p314-329, August 1988.

[7] G. R. Wright and W. R. Stevens. TCP/IP Illustrated, Volume II, The Implementation. Addison-Wesley Publishing Company, 1994.

[8] H. A. Wang and M. Schwartz, "Achieving Bounded Fairness for Multicast and TCP Traffic in the Internet". http://www.ctr.columbia.edu/~whycu/res.html.

[9] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks". Computer Networks and ISDN Systems, 17:1-14, 1989.

[10] A. Mankin, "Random drop congestion control". In Proceedings of ACM Sigcomm '90 Conference, V.20, p1-7, 1990.

[11] T. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss". IEEE/ACM Trans. Networking, 5(3):336-350, June 1997.

[12] Hashem, E, "Analysis of random drop for gateway congestion control", Report LCS TR-465, Laboratory for Computer Science, MIT, Cambridge, MA, 1989.