IMPROVING THE PERFORMANCE OF TCP VEGAS AND TCP SACK: INVESTIGATIONS AND SOLUTIONS

KRISHNAN NAIR SRIJITH
B.Eng. (First Class Hons.)

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE (COMPUTER SCIENCE)

SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2002

ACKNOWLEDGMENTS

This thesis would not have been possible without the help and support from a number of people. I am highly indebted to my advisor, A/Prof. A.L. Ananda for his invaluable support and guidance throughout my thesis work. I thank him for the enlightening conversations, clever ideas and the stimulating environment. I am also grateful for his patience and understanding when all was not well and nothing good seemed to happen.

I am thankful to Dr. Lillykutty Jacob for her support and encouragement. Her critical view of my ideas and her thoroughness in scrutinising them helped clarify and distil my thoughts.

I thank the *ns* and the Open Source community for all the help they have provided me in clarifying my questions and helping me complete this thesis successfully.

I am immensely grateful to my friends Renjish, Yuan Lihua and Yongxiang for their moral and mental support, useful inputs and patient hearing. Thanks for all the laughter we shared together for the last two years. I am grateful to Venkatesh, Saravanan and Sudarshanan for all the support they have given me.

Last but not the least, I am what I am because of the love and support of my parents, my brother and my love Surya. They have always been beside me in time of need, urging me on. Thank you!

Dedicated to my family and my love, Surya

CONTENTS

List of Figures

LIST OF TABLES

LIST OF ABBREVATION

ACK	Acknowledgment
AQM	Active Queue Management
BW	Bandwidth
BW*D	Bandwidth-Delay Product
Cwnd	Congestion Window
D-SACK	Duplicate SACK
DUPACK	Duplicate Acknowledgment
FTP	File Transfer Protocol
GEO	Geo-synchronous Earth Orbit
IP	Internet Protocol
Kb	Kilobits
KB	Kilobytes
Kbps	Kilobits per second
KBps	Kilobytes per second
Km	Kilometre
LEO	Low Earth Orbit
MSS	Maximum Segment Size
Ns	Network Simulator
PER	Packet Error Rate

RFC	Request For Comments
RTT	Round Trip Time
SACK	Selective Acknowledgment
SSthresh	Slow Start Threshold
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

SUMMARY

This thesis investigates the performance of Transmission Control Protocol (TCP) over wired and satellite networks. In particular, it concentrates on the performance of TCP Vegas congestion avoidance mechanism and TCP Selective Acknowledgement (SACK) extension to TCP. The thesis reviews existing problems associated with these protocols and proposes solutions to overcome these limitations and improve their performance. Experimental results show that the modified TCP Vegas (called Vegas-A) and modified SACK are able to solve most of the problems identified, making TCP more resilient and efficient. The major features of the proposed changes are its reduced dependence on internal protocol constants, simplicity and ability to perform well in both wired and satellite network environments.

Keywords: Transmission Control Protocol, TCP Vegas, TCP SACK, Congestion Avoidance, Protocol Fairness, Satellite network.

Chapter 1

INTRODUCTION

In the introductory chapter of the thesis, the motivation for the thesis is discussed. Then the objectives of thesis and its contribution are laid out and the thesis organization is presented.

1.1 Motivation

Transmission Control Protocol (TCP) [1] is the Internet's most widely used transport control protocol [2]. TCP's strength lies in the adaptive nature of its congestion avoidance and control algorithm and its retransmission mechanism, first proposed by V. Jacobson [3] as a part of TCP Tahoe. It was further refined in Reno [4] and New Reno [5] versions of TCP. TCP Vegas, proposed by Brakmo et al. [6] proposes a fundamentally different congestion avoidance scheme from that of TCP Reno. TCP Vegas was able to achieve up to 71% higher throughput compared to TCP Reno [6].

Even though TCP Vegas has produced outstanding results, there are a lot of unresolved issues associated with it, which has prevented it from being widely accepted and deployed over the Internet [7,8,9]. Some of the issues that have been identified so far are the unfairness experienced by TCP Vegas flows while competing with TCP New Reno flows, TCP Vegas's inability to distinguish re-routing conditions, its unfair treatment of older connections etc. It has been reported and verified [6] that TCP Vegas does perform better than New Reno in terms of throughput and packet retransmissions. In view of this, it was felt by the author that a solution to the reported

problems of Vegas would render it a better alternative to TCP New Reno implementation.

TCP SACK [10] was proposed to solve the problem that arises when New Reno loses multiple packets from a window. It is able to prevent the retransmission of received packets, thus improving the performance of TCP New Reno. However, there is a limitation on the amount of information TCP SACK can carry. Under certain conditions, TCP SACK in its present form is not able to prevent unnecessary retransmissions. The author felt that overcoming this limitation would make TCP more robust in handling packet losses and thus increase the throughput over high and erratically lossy links. Furthermore, TCP SACK can complement the working of TCP Vegas and increase the performance further.

1.2 Research Objectives

The objectives of the research described in this thesis are to:

- Study the performance of various TCP congestion avoidance, control and retransmission mechanisms over wired and satellite networks.
- Study in detail, the limitations of TCP Vegas.
- Investigate the dynamics of TCP Vegas and suggest changes to the protocol to resolve the limitations observed.
- Study the working of TCP SACK and its limitations.
- Suggest changes to the TCP SACK's current mechanism to resolve the limitations.

1.3 Thesis Contribution

As a part of the investigation into the performance of various TCP mechanisms, an extensive study was conducted on the performance of various TCP schemes over a Geosynchronous Earth Orbit (GEO) satellite network [11], as well as over a network emulator test bed. The results of the study reconfirms the various performance measurements reported in earlier studies.

The issues identified with TCP Vegas were reinvestigated and to address them, a modification to TCP Vegas's congestion avoidance algorithm is proposed. This modified algorithm TCP Vegas-A is shown to obtain a fairer share of the available bandwidth, tackle re-routing problems and solve the problems associated with older TCP Vegas flows. The experiments over simulated wired and satellite (both GEO and Lower Earth Orbit (LEO)) environment prove that TCP Vegas-A outperforms TCP Vegas and TCP New Reno under various conditions of error and congestion. The modifications proposed and the results of the experiments conducted are discussed in greater detail later in the thesis.

A change to the packet structure of TCP SACK is proposed that enables it to perform better than the present implementation of TCP SACK under certain error conditions. This decreases the number of unnecessary retransmissions of packets and thus improves the throughput of TCP connections.

1.4 Thesis Organization

The thesis is organized as follows. Chapter 2 presents a background study on TCP, TCP Vegas and TCP SACK and discusses the problems associated with each of

them. In Chapter 3 a comparison of the performance of TCP SACK and TCP New Reno over a Geo-synchronous Earth Orbit satellite link is presented. TCP Vegas's problems are analysed in Chapter 4 and a new algorithm (named TCP Vegas-A) is proposed to overcome the limitations in TCP Vegas. Simulation experiments on both wired and wireless networks are presented in this chapter to demonstrate the effectiveness of TCP Vegas-A. In Chapter 5 TCP SACK's worst-case limitation is discussed and a modified structure is proposed and demonstrated to overcome the limitations. We conclude the thesis in Chapter 6, with a summary and an indication of possible further work.

Chapter 2

BACKGROUND STUDY & KNOWN ISSUES

In this chapter a detailed study on TCP [1] is presented. We take an in-depth look at TCP Vegas and TCP SACK and examine the issues associated with these versions/extensions of TCP. We also discuss the work done by others in trying to solve some of the issues addressed and some of the shortcomings of these suggestions.

2.1 Overview

In this section we discuss the basic working of TCP, TCP SACK, TCP Vegas and working of TCP over satellite links.

2.1.1 Transmission Control Protocol (TCP)

Transmission Control Protocol is the Internet's most widely used transport control protocol [2]. TCP provides applications like FTP [12], Telnet [13] etc. with a connection oriented, reliable byte stream service on top of the Internet Protocol (IP) [14]. When a process communicates with another using TCP as the underlying transport mechanism, the sending process sends a SYN packet to which the receiving process replies with its SYN-ACK and the sender replies with an ACK. Once this three way handshake is negotiated, the connection is established and data transmission can begin. When all the data is sent, the client and the server exchange FIN and ACK in both directions and terminate the connection.

Each byte of data that is sent by a client is assigned a sequence number, unique to that session. The server acknowledges receipt of each byte of the data using ACK

segments. Acknowledgements of TCP are cumulative; an ACK confirms the successful receipt of all the data bytes up to (but not including) the acknowledged sequence number. Normally, TCP does not acknowledge each and every byte received individually, nor does it send ACK packets every time it receives data. It waits for a certain amount of time. During this period, if more data segments arrives, these segments are acknowledged together at once ("delayed acknowledgment") or if a data segment has to be sent, the acknowledgment is 'piggy backed" along with the data packet.

The major control mechanisms of TCP are its congestion avoidance and congestion control mechanism. They are discussed in detail below:

1. Slow Start

Before TCP can send data at a fast rate, it needs to estimate the bandwidth available. If this is not done, the throughput of the TCP connection will drastically decrease, as the intermediate routers would have to queue or even drop the packets for want of buffer space. The slow start mechanism adds a new parameter that controls the rate at which packets are sent, *congestion window* denoted by *cwnd*. When a new TCP connection is established, the initial value of *cwnd* is set to a value less that or equal to 2*Maximum Segment Size (MSS), but not greater than two segments [15]. Every time an ACK segment is received, the *cwnd* is increased by one segment. Thus, when an ACK arrives that acknowledges the first packet, the *cwnd* is increased to two and two data segments are sent. When ACKs for these two segments arrives, the *cwnd* is increased to four. This process thus provides an exponential increase to the *cwnd* parameter. At any point, the TCP sender can send up to the minimum of

receiver's advertised window and its own value of *cwnd*. TCP remains in this exponentially increasing slow start phase as long as *cwnd* value is less than or equal to "slow start threshold", *ssthresh* (some implementations use just the "less than" condition).

2. Congestion Avoidance

Congestion avoidance is the algorithm used by TCP to avoid losing packets and if and when packets are lost, to deal with the situation. It is described in [3]. TCP performs congestion avoidance when *cwnd* is greater than *ssthresh*. In the congestion avoidance phase, the *cwnd* is increased by 1 full-sized segment every round-trip time (RTT). Congestion avoidance continues until congestion is detected.

Congestion can be detected in two ways – receipt of duplicate acknowledgment or due to time timeout. If the detection is done using the retransmission timer timeout, the value of *ssthresh* is updated as follows:

$$\text{ssthresh} = \max(\textit{FlightSize} / 2, 2*\text{MSS})$$

FlightSize is the amount of outstanding data in the network. Then *cwnd* is set to 1 segment. After the packet has been retransmitted (whose timer had expired), TCP starts again in the slow start mode using the above mentioned algorithm to raise the *cwnd* from 1 to the new value of *ssthresh*, after which, congestion avoidance again clicks in.

However, instead of the retransmission timer expiring, if three duplicate ACKs (three duplicate ACK is equal to four identical ACKs without the arrival of any

other packet in between) were received at the sender, TCP assumes that it is an indication of packet loss. It then uses what is called the "fast retransmit" and "fast recovery" algorithm to recover the packet and fill the network again. The TCP sender does the following on the arrival of the third duplicate ACK:

- Set *ssthresh* to the value given in the equation above.
- Retransmit the lost packet and set *cwnd* to *ssthresh* + 3*MSS (fast retransmit).
- For each additional duplicate ACK that is received, *cwnd* is incremented by one MSS.
- If this new value of *cwnd* and receiver window allows, transmit new segment(s).
- When the ACK that acknowledges the receipt of a new segment is received, set *cwnd* to *ssthresh* (fast recovery).

2.1.2 TCP Vegas

TCP Vegas was proposed by Brakmo et al. in [6]. It has a very different congestion control algorithm compared to New Reno. TCP Vegas in general controls its segment flow rate based on its estimate of the available network bandwidth. Among the many new features implemented in TCP Vegas, the most important difference between it and TCP Reno lies in its bandwidth estimation scheme. Studies on TCP Vegas have shown that Vegas achieves higher efficiency than Reno, causes fewer packet retransmissions and is not biased against connections with longer RTTs [6,7,8].

TCP Vegas dynamically varies its sending window size based on fine-grained measurement of RTTs, whereas TCP Reno continues to increase its window size until

packet loss is detected. While TCP Reno views packet losses as a sign of network congestions, TCP Vegas uses a sophisticated bandwidth estimation scheme, wherein it uses the difference between the expected and achieved flow rates to estimate the available bandwidth in the network. The idea is that when the expected and actual throughputs are almost equal, the network is not congested. In other words, in a congested scenario, the actual throughput will be much smaller than the expected. Thus, based on this estimate of network congestion, TCP Vegas updates the congestion window. According to the TCP Vegas mechanism:

a) expected_rate = cwnd(t)/base_rtt

   where cwnd(t) is the current congestion window size and base_rtt is the minimum RTT of that connection.

b) actual_rate = cwnd(t)/rtt

   where rtt is the present round-trip time

c) The source estimates the backlog in the router queue from the difference

   diff = expected_rate – actual_rate

d) Using this value of diff, the congestion window value (*cwnd) is adjusted as:*

$$cwnd = \begin{cases} cwnd + 1 \text{ if } diff < \alpha \\ cwnd - 1 \text{ if } diff > \beta \\ cwnd \quad otherwise \end{cases}$$

The algorithm works on the understanding that when the expected and the actual throughput are close to each other, the connection may not be utilizing the available network bandwidth, and hence should increase the flow rate. On the other hand, when the actual throughput is much less than the expected throughput, the network is probably experiencing congestion and hence the connection should reduce the flow rate.

2.1.3 The Selective Acknowledgment extension to TCP

New Reno is the default TCP implementation in most systems. In this implementation, the sender can recover from an isolated packet loss using the fast retransmit and fast recovery algorithms mentioned earlier. However, when multiple packets are lost from a single window, the sender will end up either retransmitting packets that have been successfully received or retransmit at most one packet per RTT [16]. TCP Selective Acknowledgment (SACK) option was introduced in RFC 2018 [10] and later extended in RFC 2883 [17] to overcome this limitation. In TCP SACK, the receiver can inform the sender about the packets that have been received successfully, thus allowing the sender to selectively retransmit the lost packets alone.

The SACK extension uses two TCP options. The first option called "SACK-permitted" is used to enable SACK in the connection. It is set along with the SYN flag. The other option used is the SACK option itself, which can be sent during a SACK enabled TCP connection. The structure of the SACK-permitted option is as shown in Figure 1, while Figure 2 shows the SACK option format.

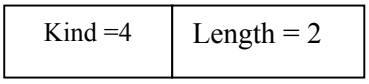

Fig. 1. TCP SACK-permitted Option

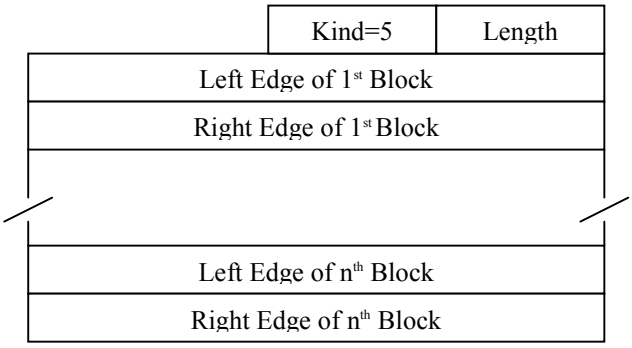

Fig. 2. Current SACK Option format

The SACK option is sent by the data receiver to inform the sender of the receipt of non-contiguous blocks of data. Contiguous block of data in SACK is represented by two 32 bit unsigned integers that form the left edge (first sequence number of the block) and the right edge of the block (sequence number immediately following the last sequence number of the block). The first SACK block specifies the contiguous block that triggered this ACK. The rest of the SACK blocks are filled by repeating the most recently reported SACK blocks (first SACK block in the previous SACK options), which does not form a subset of any of the SACK block already included in the present SACK option.

A SACK option will thus have a length of 8*n+2, where n is the number of SACK blocks included. Thus, if no other TCP options are enabled, the 40 bytes of TCP header can hold at the most 4 SACK blocks.

2.1.4 TCP over Satellite networks

Several companies have announced plans to build large satellite systems to provide commercial broadband data service. However, the performance of data communication protocols and applications over such systems is the subject of heated debate in the research community. Nowhere is it more raged than in discussions regarding TCP. In the recent few years, there have been several experimental studies on the efficacy of using standard TCP/IP protocols over satellite channels. Allman and Kruse [18] present several references to those studies as well as the main results. There are several IETF standardized mechanisms [19] as well as a number of other possible TCP versions [20] that may allow TCP to better utilize the available bandwidth provided by networks containing satellite links.

2.2 Issues

Several issues have been identified with TCP Vegas, TCP SACK and the general performance of TCP over satellite links. In this section of the chapter we look at these issues in detail.

2.2.1 Issues with TCP Vegas

Although it has been shown that TCP Vegas can provide better performance in terms of increased throughput, reduced jitter and much reduced retransmissions, there are a number of issues associated with TCP Vegas that prevent it from being accepted and widely deployed in the Internet.

A. *Fairness*

TCP Vegas uses a conservative algorithm to decide when to vary the congestion window. However, TCP Reno, in an effort to fully utilize the bandwidth, continues to increase the window size until a packet loss is detected. Thus, while TCP Vegas tries to maintain a smaller queue, TCP Reno and its newer variant New Reno, keeps a larger number of packets in the buffer on an average, and thus steals a larger share of the available bandwidth.

When TCP Vegas and TCP Reno connections share a bottleneck link, Reno uses up most of the available router buffer space. TCP Vegas, interpreting this as a sign of congestion, decreases the congestion window, leading to a situation where New Reno enjoys more than the fair share of bandwidth. Furthermore, when the router buffer size is large, the average window size of TCP Reno connections will be large, while the window size of TCP Vegas connections will remain unchanged, as it does

not inflate the window size larger than β. This means that the larger the router buffer size, the worse the fairness between Reno and Vegas connections.

Hasegawa et al. [21] proposed TCP Vegas$^+$ as a method to tackle TCP Vegas's fairness issue. Vegas$^+$ has two modes. In the moderate mode, Vegas$^+$ behaves identically to the normal TCP Vegas. But it enters the aggressive mode to increase its window size aggressively (identical to TCP Reno) when it detects the presence of competing Reno traffic. However, Vegas$^+$ assumes that an increase in the RTT value is always due to the presence of competing traffic and rules out other possibilities like re-routing. Furthermore, the performance of Vegas$^+$ depends on the choice of optimal value for the variable $count_{max}$ in order to switch between the two modes.

B. *Re-routing*

In Vegas, the parameter *baseRTT* denotes the smallest round-trip delay and is used to measure the expected throughput. When the route is changed, the RTT of a connection can change. Vegas is not usually affected if the new route has a smaller RTT, since *baseRTT* will be updated dynamically. But when the new route has a longer RTT, the connection will not be able to deduce whether the longer RTTs experienced are because of congestion or because of route change. Without this knowledge, TCP Vegas assumes that the increase in RTT is because of congestion along the network path and hence decreases the congestion window size.

This is exactly the opposite of what the connection should be doing. When the propagation delay increases, the bandwidth-delay product (bw*d), which is an estimate of the number of packets that can be held in the network, increases. For any

connection, the expression (cwnd – bw*d) gives the number of packets in the buffers of the routers. As the aim of TCP Vegas is to keep the number of packets in the router between the values of α and β, it should increase the congestion window to keep the same number of packets in the buffer when the propagations delay increases.

La et al. [9] proposed a modification to Vegas to counteract the re-routing problem. Their modification assumes that any lasting increase in RTT is a sign of re-routing. For the first K packets, the mechanism is the same as TCP Vegas. However, subsequently, the source keeps track of the minimum RTT of every N packets. If this minimum RTT is much larger than the *baseRTT* for L consecutive times, the source updates its *baseRTT* to the minimum RTT of the last N packets and resets the cwnd based on this new *baseRTT*. Their reasoning for this modification is that since the increase in RTT forces the source to decrease the cwnd, the increase in RTT comes mostly from the propagation delay of the new route and not from the congestion. However, this mechanism adds two more parameters δ and γ that controls the update process of *baseRTT*, and finding the appropriate values for K, N, L, δ and γ remains an open issue.

C. *Unfair treatment of 'old' connections*

As pointed out in [22], TCP Vegas's congestion control mechanism is inherently unfair to older connections. When a Vegas connection is established in an uncongested network, the *baseRTT* is likely to be close to the minimal RTT possible. When, later on, the network gets congested, the measured RTT rtt_1 increases and the ratio $baseRTT_1/rtt_1$ decreases. Consider a new connection that is initiated after the network becomes congested. The value of $baseRTT_2$ measured for this connection will

be larger than $baseRTT_1$. Hence $baseRTT_2/rtt_2$ will be larger than $baseRTT_1/rtt_1$ as rtt_1 and rtt_2 are nearly equal. As derived in [22], window size that triggers a reduction in congestion window size is given by:

$$\text{cwnd} > \frac{\beta}{1-\frac{baseRTT}{rtt}}$$

Thus, we see that for the older connection, the critical value of cwnd is smaller than that of the newer connection. Hence, the second connection can achieve higher bandwidth than the first (older) connection.

Similarly, the critical value of congestion window that triggers an increase in congestion window is given by:

$$\text{cwnd} < \frac{\alpha}{1-\frac{baseRTT}{rtt}}$$

As the right side hand side term is larger for the newer connection, it is more likely to be able to increase its congestion window than the older connection, and thus again get more than a fair share of the network bandwidth.

A closely related problem is the persistent congestion problem, where the connections may overestimate the propagation delays and possibly drive the network to a persistently congested state. Consider a connection that starts when the network is congested. As of result of the queuing delay caused by the backlog from other connections, the packets from the new connection experience RTTs that are considerably larger than the actual propagation delay of the path in the uncongested state. Thus, because of this inaccurate estimation of the baseRTT, it will overestimate the possible window size (as pointed out by the critical window sizes that trigger

increase/decrease of the cwnd). This scenario will repeat for each new connection, leading to persistent congestion. La et al. [9] suggested that the same solution as the one they proposed for the re-routing problem, together with RED enabled routers could solve the persistent congestion problem. Low et al. [23] also proposed augmenting Vegas with appropriate active queue management (AQM) such as Random Exponential Marking (REM) for avoiding the persistent congestion problem.

2.2.2 Issues with TCP SACK

In [24], Floyd addressed various issues related to the behavior and performance of TCP SACK. One important limitation mentioned in [24] and in RFC 2018 [10] is that TCP SACK requires 64 bits (8 bytes) to represent the upper and lower bound sequence numbers of every block it selectively acknowledges. Since TCP protocol limits the maximum length of the options field to 40 bytes and SACK is usually implemented along with the TCP Timestamp option, an acknowledgment packet can carry a maximum of only three blocks' information.

In the extension proposed to SACK in RFC 2883 [17], a new mechanism called DSACK was proposed wherein the first block of SACK is used to carry information about the latest duplicate packet received. This further reduces the amount of actual SACK information that can be carried in the ACK packets. Similarly, if other TCP options are enabled, this maximum number would decrease further. Usage of more TCP options will become more common as more and more options to TCP are being put forward and accepted.

This rather small maximum number of SACK block information can lead to efficiency problems in the performance of TCP. Under certain packet loss scenarios, the TCP sender ends up retransmitting packets that have already been received successfully. An example of such a situation is described in [24], which is reproduced here, as it forms an interesting scenario for our simulations. In this scenario, it is assumed that the congestion window is at least 11 packets large, with a loss of at least four data and three ACK packets. Table 1 describes the scenario in more detail.

Data Packets	ACK Packets	Sender reaction
1	Normal ACK –1	Send 12
2 (lost)		
3	Dup ACK-1,3-3	No action
4	Dup ACK-1,3-4	No action
5	Dup ACK-1,3-5	Retrx pkt. 2
6	Dup ACK-1,3-6 (lost)	
7 (lost)		
8	Dup ACK-1,8-8, 3-6 (lost)	
9 (lost)		
10	Dup ACK-1,10-10,8-8, 3-6 (lost)	
11 (lost)		
12	Dup ACK-1,12-12,10-10,8-8	Retrx pkt. 6

Table 1. SACK error scenario

As shown in the table, data packets 2,7,9 and 11 are lost. ACKs of data packets 6,8 and 10 are also lost. As a result, data packet 6 is retransmitted unnecessarily. This is because the duplicate ACK that was sent when packet 12 was received did not have enough space to carry the information that packet 6 had already been received (it contained only SACK blocks 12-12, 10-10 and 8-8 and not 3-6). Had this information been conveyed, the unnecessary retransmission of packet 6 could have been avoided.

2.2.3 Problems with TCP over Satellite links

A number of characteristics of satellite links may degrade the performance of TCP over it. The important ones among them are discussed below.

A. Long RTT

Satellite links have an average RTT of around 500ms. TCP uses the slow start mechanism to probe the network at the start of a connection. Time spent in slow start stage is directly proportional to the round trip time (RTT) and for a satellite link, it means that TCP stays in slow start mode for a longer time than in the case of a small RTT link. This drastically decreases the throughput of a short duration TCP connection. Furthermore, when packets are lost, TCP enters the congestion control phase, and due to higher RTT, remains in this phase for a longer time, thus reducing the throughput of both short and long duration TCP connections.

B. Large Bandwidth-Delay product

Bandwidth-Delay product is a measure of the amount of data in flight on a link at any point of time. The high RTT of the satellite increases the bandwidth-delay product. In windowed protocols like TCP, the value of the bandwidth-delay product is very critical. To fully utilize the link, the window size of the connection should be equal to the bandwidth-delay product.

In TCP, without the window scaling option, since the TCP header uses a `6 bit field to report the receive window size to the sender; the largest window size possible is 64KB. For a GEO satellite the maximum throughput achievable using this window size is:

$$throughput_{\max(satellite)} = \frac{advertised\ window}{round\ trip\ time} \approx \frac{64KB}{500ms} \approx 1024Kbps$$

Thus, a TCP connection, which might be using a satellite link of 2Mbps bandwidth, ends up achieving a maximum throughput of just over 1Mbps. In addition,

many TCP stack implementations use advertised window sizes much less than 64KB by default. To remedy this, RFC 1323 [25] specifies that TCP may use a maximum window size of up to 2^{30}, by using the window scaling option. This extension expands the definition of TCP window to 32 bits and then uses a scale factor to carry this 32-bit value in the 16-bit window field of TCP header.

C. *Link Errors*

One of the biggest drawbacks of TCP is that it cannot distinguish between packet loss because of link error and that caused by link congestion. Any segment lost is always considered as being caused by congestion and TCP performs congestion avoidance with the receipt of three duplicate ACKs or slow start in the case of timeout. As mentioned before, because of the long RTT value, once it enters any of these states, TCP connections on satellite links take a longer time to return to the throughput level that the connection was enjoying just before entering the congestion control phase. Thus errors on a satellite link have a more deleterious effect on the performance of TCP rather than over low latency links.

2.3 Summary

In this chapter, an introduction to TCP and its congestion control and congestion avoidance algorithms were presented. Then TCP Vegas was looked at and limitations associated with it were reviewed. The TCP SACK mechanism was introduced next and issues associated with it were examined. The chapter ended with an overview of problems associated with TCP performance on satellite links.

Chapter 3

EFFECTIVENESS OF TCP VARIATIONS OVER SATELLITE LINKS

This chapter reports the study on the comparison of performance of TCP SACK and TCP New Reno over a geosynchronous earth orbit (GEO) Satellite link. It is shown that SACK enabled TCP connections have a better throughput compared to normal TCP New Reno connections.

3.1 Experimental Setup

To analyze the performance of TCP New Reno and SACK, a variety of experiments were conducted, using both a link emulator as well as the GEO satellite link. The link emulator was used because of the limited testing time allowed with the satellite link and also because of the flexibility in simulating error scenarios. Intel Pentium machines running the Linux 2.2.14-5.0 kernel were used for the experiments. *Iperf* [26] was used to generate TCP and UDP traffic. *Tcpdump* [27] was used to observe the TCP packets and *tcptrace* [28] was used to analyze the output from *tcpdump*. The experimental testbed using the link emulator is show in Figure 3 and the testbed using the GEO satellite link is shown in Figure 4. The error/delay box in Figure 3 used a modified version of *rshaper* [29] to corrupt and delay packets in the network to simulate lossy and long latency environments, respectively. Various conditions of corruption and congestion were inserted/simulated in the satellite link to review the performance variation of SACK and New Reno.

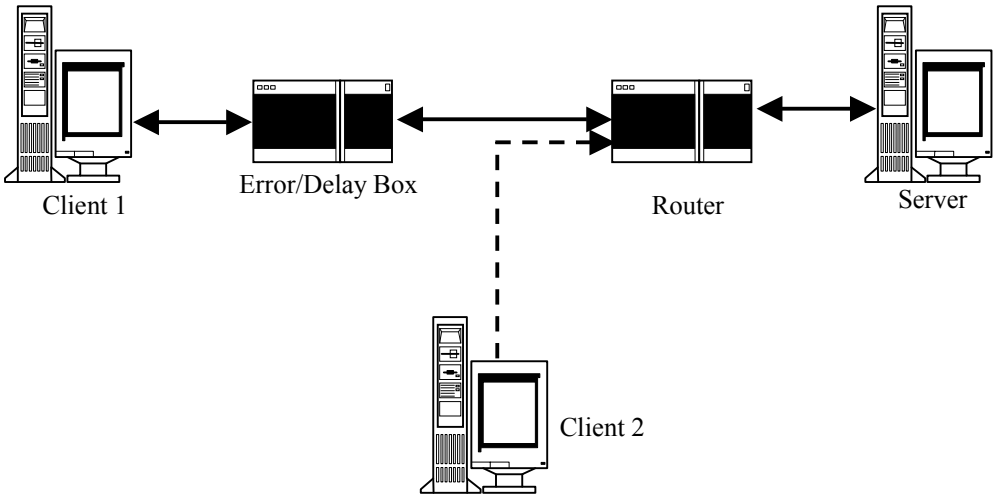

Fig. 3. Experiment testbed 1

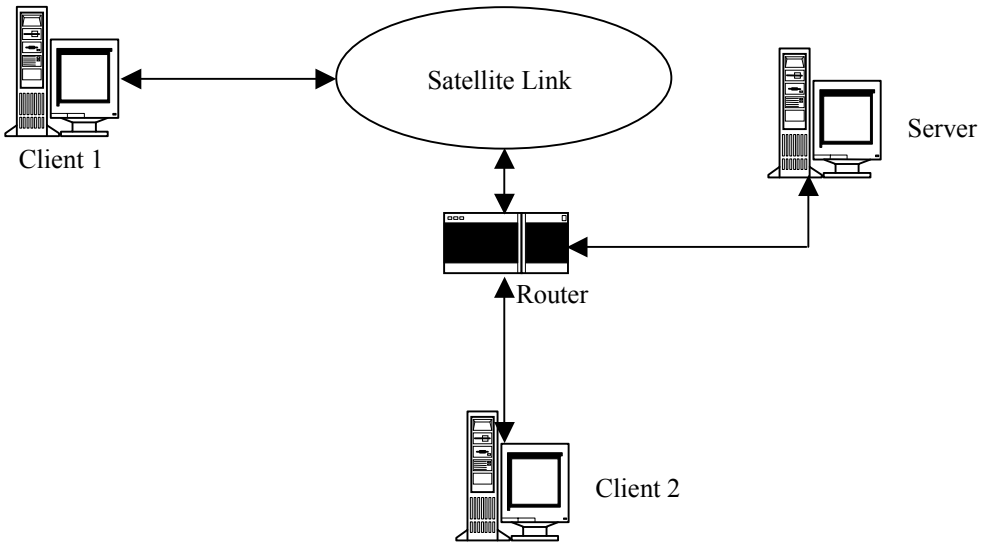

Fig. 4. Experiment testbed 2

3.2 Results and Discussion

3.2.1 Link Emulator with No Corruption

The first set of experiments tested the performance of TCP New Reno and SACK over the long latency link in a no-corruption environment. An RTT of 510 ms was introduced by the error/delay box to simulate the long latency of the satellite link. Files of different sizes (100KB, 1MB and 10MB) were sent from client 1 to server (Figure 3). The TCP window size was also varied from 32KB to 1024KB. At the server, the goodput• obtained was measured.

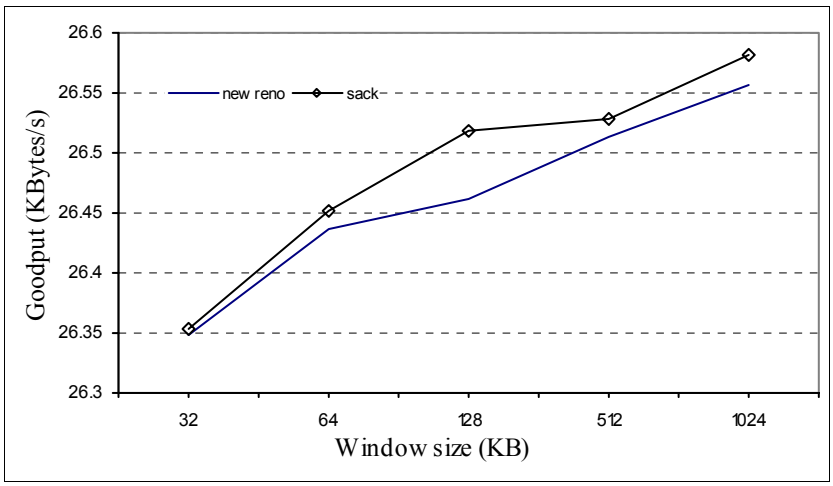

Fig. 5. Goodput for 100KB file transfer for different window sizes - no corruption

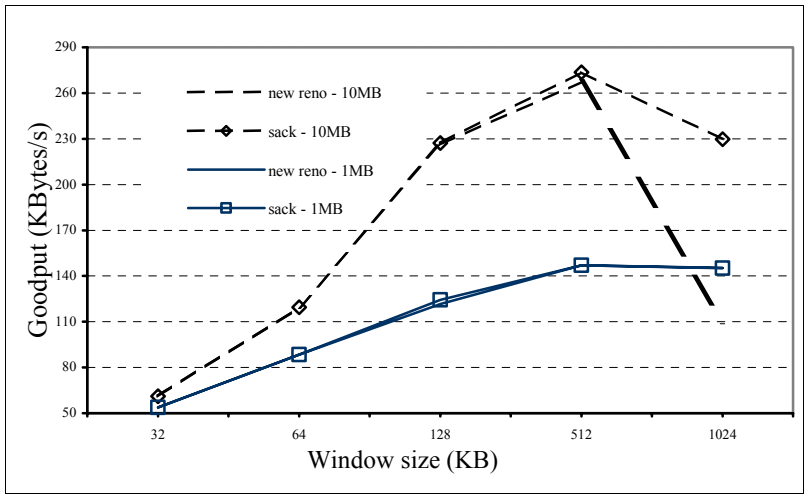

Fig. 6. Goodput for 1MB and 10MB file transfers for different window sizes - no corruption

• Defined as the effective bandwidth delivered to the receiver, excluding overheads and duplicate packets caused by retransmissions.

It can be seen from Figures 5 and 6 that when the window size is increased, the goodput generally increases, except for the case with 10MB file and window size of 1024KB. The result is consistent with earlier published results and endorses the TCP window scaling option in RFC 1323 [25]. It is interesting to see that the goodput for 100KB and 1MB file transfers are smaller than that for the 10 MB file transfer for all window sizes. This is because in both the cases, there is not enough data to actually take advantage of the large TCP window size and fill the link.

The performances of New Reno and SACK are comparable except at very large window size. The comparable performances of New Reno and SACK are expected because of the no-loss environment. However, for a large window size of 1024KB, for the 10MB file transfer (Figure 6), the goodput decreases in both cases, but more in the New Reno case. This can be explained as follows. For a link with RTT of 510 ms and bandwidth of 10Mbps, the bandwidth-delay product is 652.8KB. When the window size is larger than this value (1024KB in this case), congestion sets in and the throughput falls. As SACK is able to handle dropped packets by using selective ACKs, it fares better compared to New Reno. Thus we see that blindly increasing window size to a very large value will actually adversely affect the throughput, even if SACK is used.

3.2.2 Link Emulator with Corruption

In the next set of experiments, packet errors were introduced into the link using the delay/error box. The packet corruption rate was set to 1% and the RTT remained 510ms. Note that 1% packet error rate corresponds to about 0.9×10^{-6} bit error rate (BER) for a segment size of 1460 bytes. File transfers of 1MB and 10MB were carried

out with varying window sizes. Figure 7 shows the resulting goodputs that were obtained.

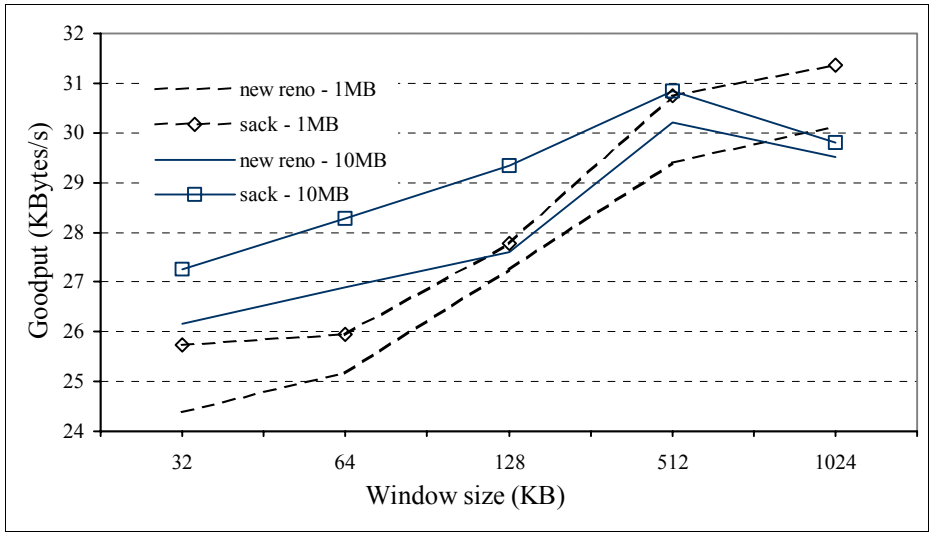

Fig. 7. Goodput for 1MB and 10MB file transfers for different window sizes - 1% corruption

The above graphs show that SACK provides better performance compared to New Reno when link experiences corruption. As with the previous case of no corruption, the 10MB file transfer goodput decreases when window size is increased beyond 652.8KB because of the presence of congestion, in addition to corruption. SACK is able to handle this situation better and provides a better goodput. The percentage reduction in goodput from no corruption to corruption case is more for 10MB file transfer because of the ten times larger number of packets loss events and consequent reduction in the TCP congestion window during the transfer.

The above experiment was repeated with packet corruption rates of 0.5% and 2%. Figure 8 summarizes the results observed for the 10MB file transfer.

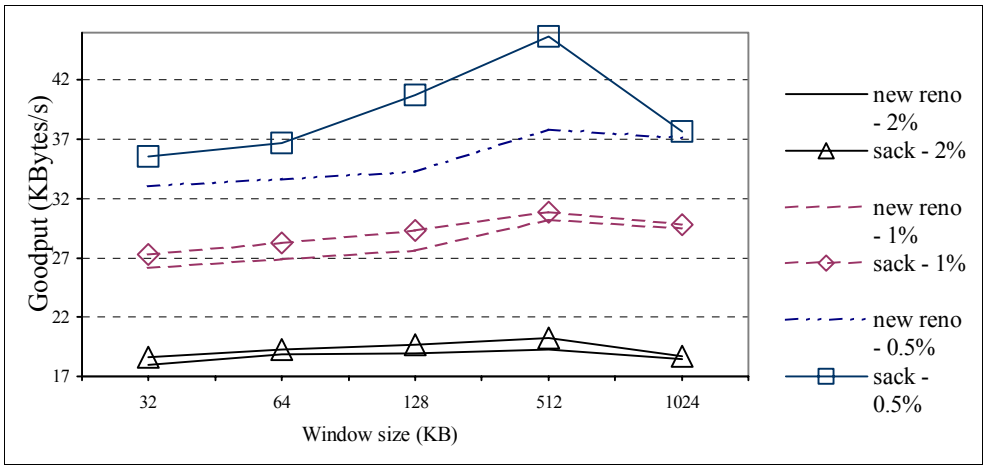

Fig. 8. Goodput for a 10MB file transfer for different window sizes and corruption rates

The result shows that SACK performs better than New Reno for all corruption levels. However, we see that the margin of improvement is greatest for lower corruption rates. As corruption rate increases, the difference in throughput between SACK and New Reno decreases, especially for very large window sizes. For the largest window size used, 1024KB, the difference between the goodput (in bytes/sec) for different corruption rates is as follows:

Corruption	SACK	New Reno	Difference
0.5 %	37055	37653	598
1.0 %	29509	29798	289
2.0 %	18463	18684	221

Table 2. Goodput for 10MB file transfer using 1024KB window

3.2.3 Link Emulator – Congestion and Corruption

In the experiments so far, onset of congestion was noticed when the window size was larger than the delay-bandwidth product. In the next experiment we introduced congestion using non-responsive UDP flows. The setup was such that only TCP data was corrupted. The corruption rate was set to 0% (no corruption, only congestion) and 1%. The window size was fixed at 512KB so that we could take full advantage of a larger window without running the risk of creating additional link congestion. The file size used was 10MB. The results for 1% corruption are shown in Figure 9. Similar behaviour was observed for 0% corruption, but of course with higher goodput values.

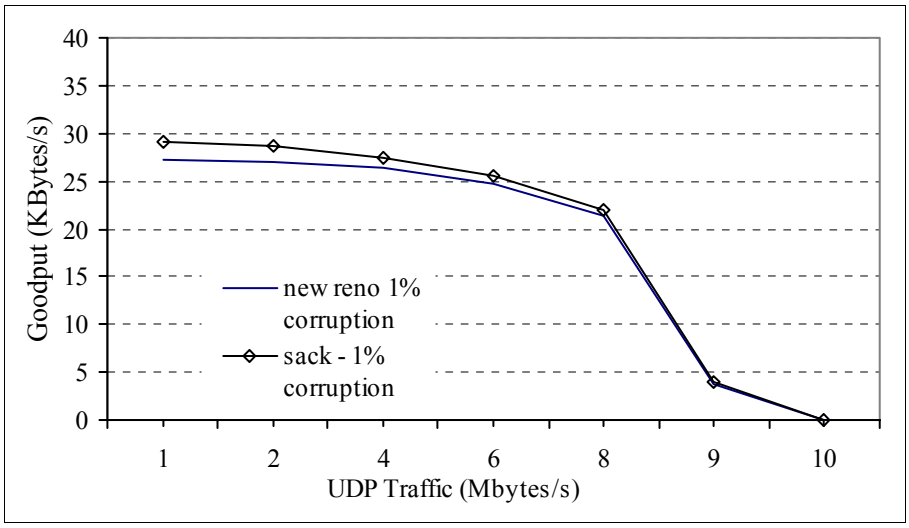

Fig. 9. Goodput for 10MB file transfer –
1% corruption and background UDP traffic

In this congestion scenario, with both 0% and 1% packet corruption rates, SACK again provided better goodput compared to New Reno when the combined effect of corruption and congestion was moderate. When the combined effect of corruption and congestion was high, SACK was not able to recover fast enough and the throughput decreased. Notice that when 10MBytes/s UDP was pumped into the

network, TCP connection was killed, as expected, because the non-responsive UDP traffic was taking up the whole link bandwidth of 10Mbps.

3.2.4 Satellite Link

Here the performance of TCP New Reno and SACK over the satellite link in Figure 4 is discussed. The satellite link had a bandwidth of 2Mbps and an RTT of 510ms. 1MB and 10MB files were transferred from client 1 to server. Window size was varied from 64KB to 128KB and finally to 256KB. Each experiment was repeated five times and the average value was taken for the final analysis. Table 3 presents the goodputs measured in KBytes/s.

Goodput → / Window size ↓	1MB New Reno	1MB SACK	10MB New Reno	10MB SACK
64KB	13	14	16.5	17.6
128KB	13.75	15	16.5	18.5
256KB	12.5	13	15.75	17.75

Table 3. Goodput for 1MB and 10MB file transfers for varying window sizes on the satellite link

The results are similar to those obtained using the link emulator. The goodput increases when the window size is increased, as long as the window size is kept below the bandwidth-delay product value (2Mbps*510ms =130.56KB). When the window size is set to 256KB, we notice that the throughput decreases for both New Reno and SACK. This is consistent with the results obtained earlier when using the link emulator. The results also show that SACK performs better than New Reno for both the file sizes as well as for all the window sizes used. This too is consistent with the results of the previous experiments. For the sake of completeness, it has to be

mentioned that these deductions are based on the limited number of experimental runs and hence the results cannot be stated with the highest level of confidence.

3.3 Summary

In this chapter, the performance of TCP SACK extension was compared with the TCP New Reno version in a GEO satellite environment. It was shown by experimental observation that TCP SACK does perform better than New Reno in long latency, error and congestion prone environment. However, there is a limit to which SACK can be helpful. It was shown that SACK has no effect when the level of corruption or congestion is very high.

Chapter 4

TCP VEGAS-A: SOLVING ISSUES WITH TCP VEGAS

In this chapter, modifications to the congestion avoidance mechanism of TCP Vegas are proposed to solve the issues identified with it in Chapter 3. Simulations are carried out to test these modifications and their results are presented. The results show that TCP Vegas-A does mitigate most of the problems associated with TCP Vegas.

4.1 TCP Vegas-A Algorithm

TCP Vegas has fixed values for the parameters α and β, usually set to 1 and 3. Recall that the Vegas strategy is to adjust the source's sending rate (congestion window) in an attempt to keep a small number of packets buffered in the routers along the path. Thus, when Vegas connections compete with Reno connections, the fairness problem is worse with larger router buffers. Since the average number of packets in the router buffer is to be kept within α and β, the main idea of the modified algorithm Vegas-A, ('A' stands for "Adaptive") is that rather than fixing α and β, they be made adaptive. At the start of a connection, α is set to 1 and β to 3. These values however change dynamically depending on the network conditions. A detailed description of the mechanism is given below.

The slowstart and congestion recovery algorithms of Vegas-A are the same as that of Vegas. However, Vegas-A uses the modified congestion avoidance mechanism:

Terms used:

$Th_{(t)}$ = actual throughput at time t

$Th_{(t-rtt)}$ = actual throughput at previous rtt

The definitions of expected_rate, actual_rate and diff are the same as those in Vegas.

```
01          if β > diff > α {
02                  if Th(t) > Th(t-rtt)  {
03                          cwnd = cwnd +1
04                          α= α+1, β= β+1
05                  }
06                  else if Th(t) <= Th(t-rtt) {
07                          no update of cwnd, α, β
08                  }
09          }
10          else if diff < α {
11                  if α >1 and Th(t) > Th(t-rtt) {
12                          cwnd = cwnd +1
13                  }
14                  else if α >1 and Th(t) < Th(t-rtt) {
15                          cwnd = cwnd –1, α= α-1, β= β-1
16                  }
17                  else if α ==1
18                          cwnd  = cwnd+1
19          }
20          else if diff > β {
21                  cwnd= cwnd-1, α= α-1, β= β-1
22          }
23          else {
24                  no update of cwnd, α, β
25          }
```

Explanation for line 3 - Even though diff > α, the throughput has been increasing. This indicates that the network is not fully utilized and that network bandwidth is still available. Hence, the sending rate can be increased, to probe the network.

Explanation for line 4 - As throughput is increasing over time, diff is decreasing. The existing small values of α and β are preventing the connection from making use of the available bandwidth. Hence α and β are increased to help congestion window grow. In the present implementation, α and β are increased and decreased at the same time to maintain their relationship with each other, as in the original implementation.

Explanation for line 15 - In Vegas-A, a small value of diff needs not necessarily imply that the bandwidth utilization is poor. It might be that the dynamically changing value of α has grown to a large value and when congestion occurs, even a small throughput can still make diff less than α. Hence cwnd and the inflated α and β needs to be decreased.

4.2 Simulation Details

This section describes the simulations done to demonstrate the effectiveness of TCP Vegas-A in solving the problems associated with TCP Vegas. The simulations were done on wired and satellite environments (GEO and LEO). *Network Simulator* 2 (NS 2) [30] was used for the simulations. The "TCP/Vegas" agent implemented in NS 2 (which is based on USC's NetBSD Vegas implementation) was modified as per the algorithm in Section 4.1 to implement the Vegas-A agent.

4.2.1 Vegas-A in wired environment

I. *Network Topology*

The generic network topology simulated is as shown in Figure 10.

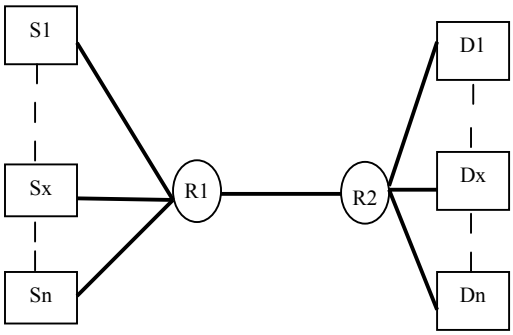

Fig. 10. Simulated wired network topology

The data rates and propagation delays of the various links - source-to-router R1, R1-to-R2 (the bottleneck link), and R2-to-destination - are specified in the context of each experiment. Routers use drop-tail policy with a buffer size of 50 packets, unless stated otherwise.

II. *Simulation Results and Discussion*

A. Re-routing

In this scenario, a route change in Vegas's connection (S1- D1) was simulated by changing the RTT of the link connecting S1 to R1. The link S1-R1 had a bandwidth of 1Mbps and initial RTT of 20ms. After 20 seconds of file transfer over the connection, the RTT of the link is changed to 200ms. The links R1-R2 and R2-D1 had a bandwidth of 1Mbps each and an RTT of 10ms for the entire duration of the connection. The simulation was run for 200 seconds to give the connection time to stabilise.

Table 4 shows the comparison of average throughputs (in bits/sec) obtained in the two cases.

	TCP Vegas	TCP Vegas-A	Difference	% Increase
Throughput	217320	940240	772920	333 %

Table 4. Throughputs for re-routing condition

Figures 11 and 12 show the variation of throughputs for the full run. The dotted curve denotes the instantaneous throughput, averaged over 0.1 second intervals, and the solid one denotes the average throughput. As soon as the RTT changes at 20 seconds, the instantaneous throughputs of Vegas and Vegas-A start to decrease. However, the throughput of Vegas (see Figure 11) went all the way down to 0.12Mbps. As seen in Figure 12, the throughput of Vegas-A does suffer initially when the RTT changes at 20 seconds. The "actual throughput" decreases, *diff* grows and hence, *cwnd* decreases. But when *cwnd* decreases, 'expected throughput' also decreases. Finally *cwnd* decreases so much so that *diff* falls between α and β.

Then, when there is a slight improvement in the throughput for any particular RTT cycle, the Vegas-A connection increases the value of *cwnd*, α and β. This increase in *cwnd* in turn increases the throughput further, which then triggers an increase in *cwnd*, α and β again. This process continues until *diff* becomes less than α. Then *cwnd* alone increases. This growth of the congestion window is shown in Figure 13. The end result is that the bandwidth available is fully utilized and the throughput goes back to the same large value as what it was prior to the re-routing. Figure 14 shows the throughput variation for TCP New Reno under similar conditions.

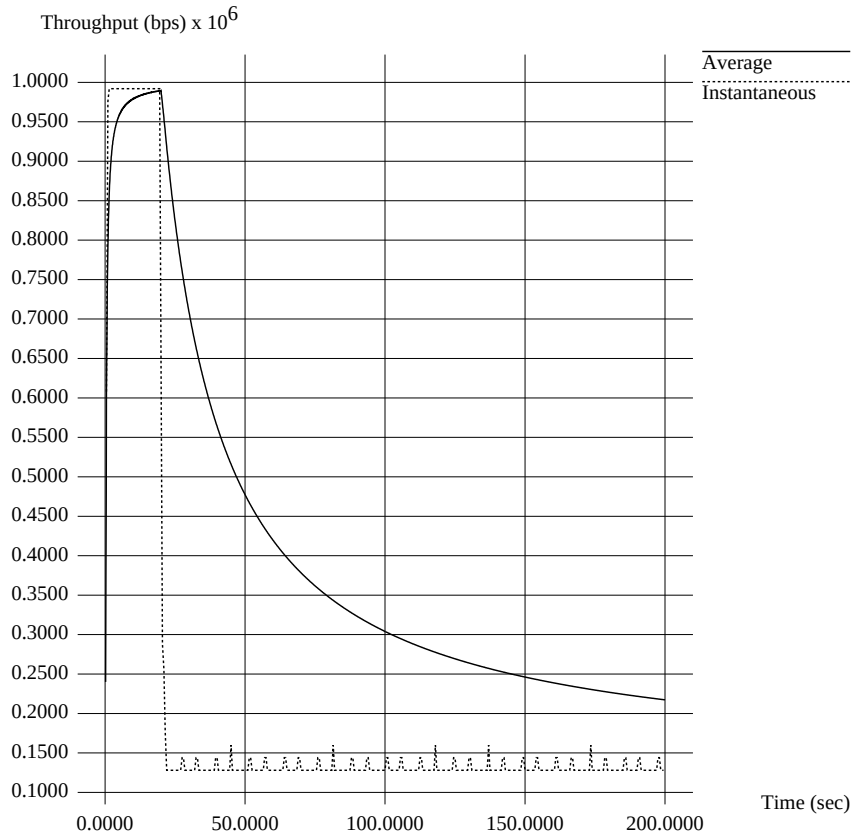

Fig. 11. Throughput variation for Vegas as a result of RTT change

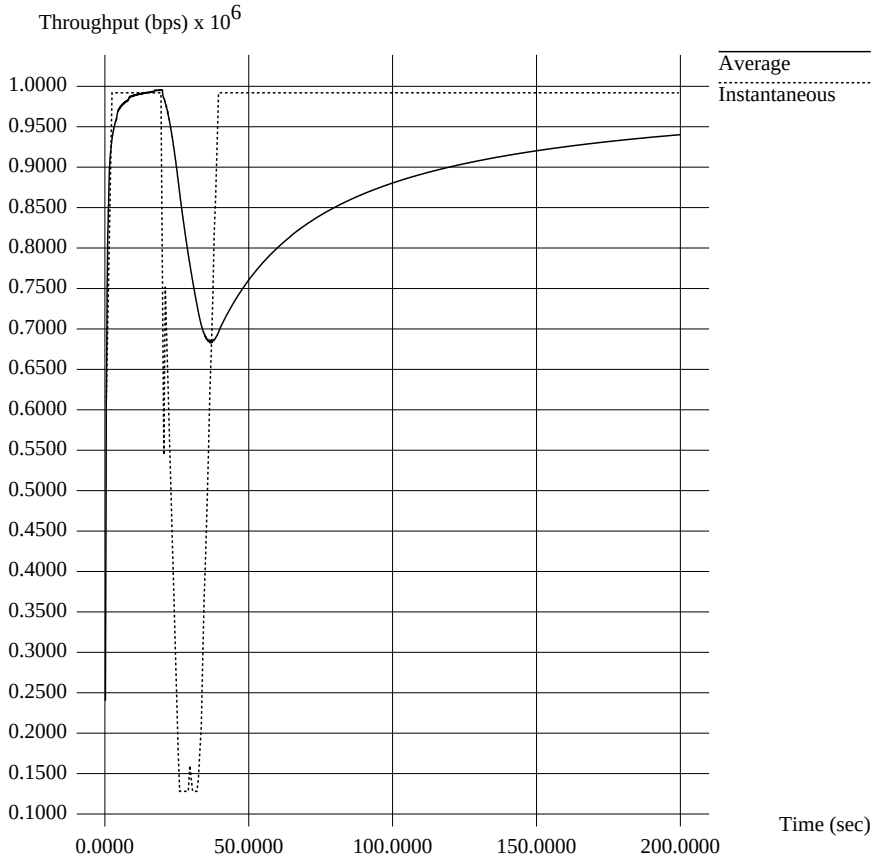

Fig. 12. Throughput variation for Vegas-A as a result of RTT change

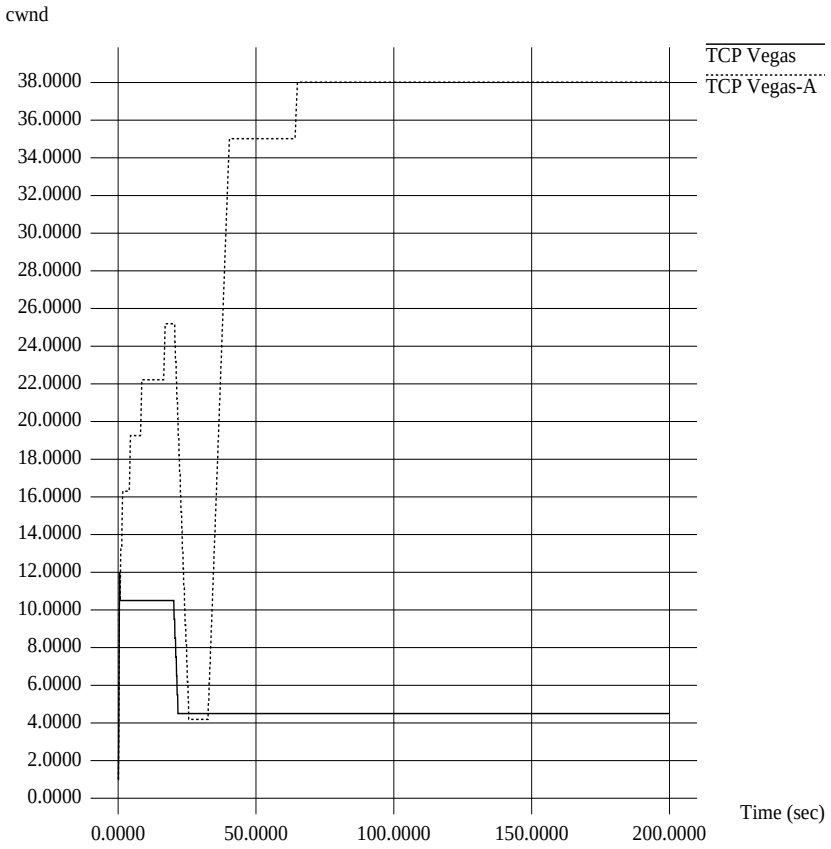

Fig. 13. *cwnd* variation for Vegas and Vegas-A as a result of RTT change

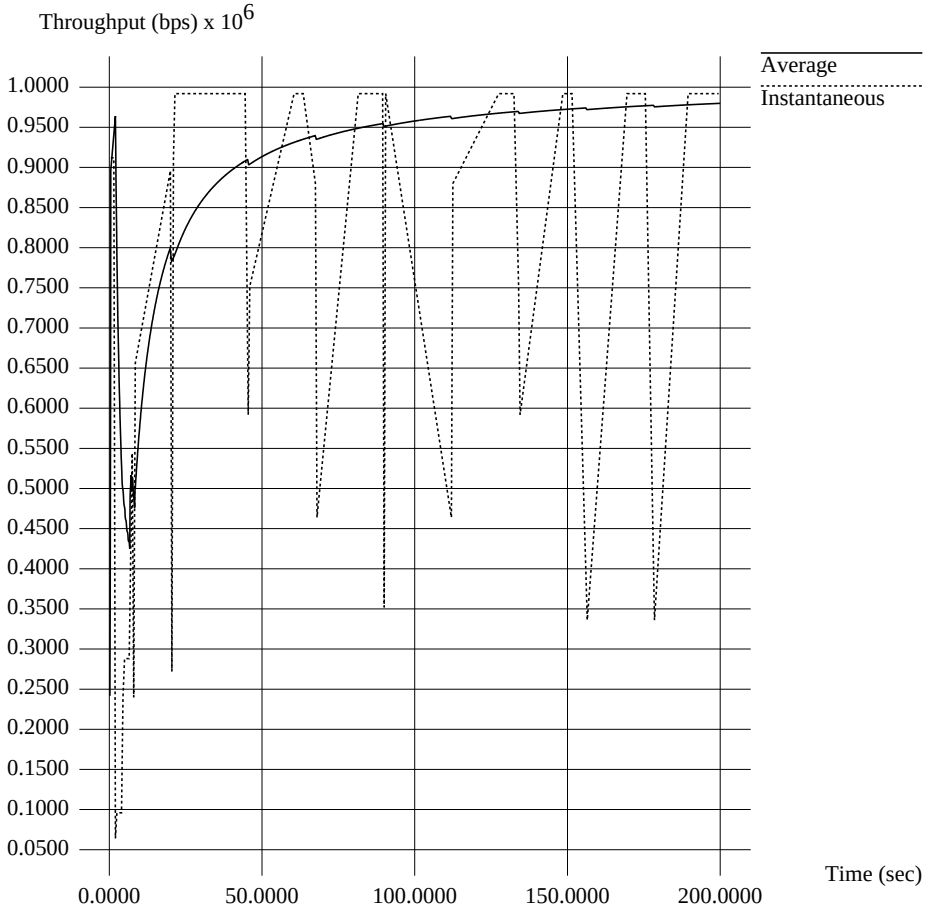

Fig. 14. Throughput variation for New Reno as a result of RTT change

B. Bandwidth sharing with New Reno

Next, the performance of Vegas-A when it shares a link with TCP New Reno is considered. There are two connections, S1-D1 and S2-D2 whose sources transfer files, but the former one uses TCP Vegas-A or Vegas and the latter one uses TCP New Reno. The links S1-R1 and S2-R1 are both 8Mbps with RTT of 20ms. The bottleneck link R1-R2 is of bandwidth 800Kbps and RTT of 80ms. R2-D1 and R2-D2 links are both 8Mbps and RTT 20 ms. S1 was started first and then S2's New Reno traffic was introduced after 10 seconds. It was found that changing the order of connections does not change the trend in the results.

Figure 15 shows the instantaneous and average throughputs of the TCP New Reno and TCP Vegas connections as they compete with each other for the bandwidth. As can be seen from the figure, when TCP New Reno starts at 10 seconds, it starts consuming Vegas's share of bandwidth.

However, when TCP Vegas is replaced with Vegas-A, the results are very different, as seen in Figure 16. Even though TCP New Reno still gets a more than fair share of the bandwidth, the unfairness is not as much as in the case of TCP Vegas. Table 5 gives numerical value of the throughput of each connection. The use of Vegas-A reduced the 'unfairness' from a ratio of 5.3:1 to 3.2:1.

$\frac{\text{New Reno}}{\text{Vegas}}$	$\frac{\text{New Reno}}{\text{Vegas - A}}$
$\frac{703875}{132040} = 5.33$	$\frac{633307}{199320} = 3.17$

Table 5. Throughput ratios for New Reno-Vegas and New Reno-Vegas-A connections

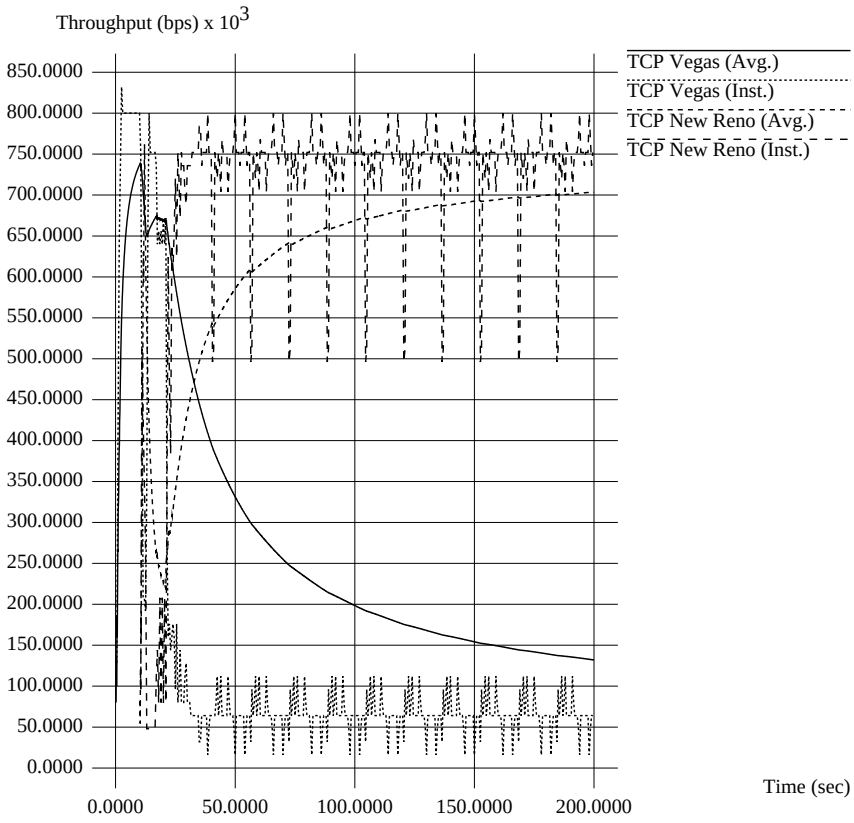

Fig. 15. Throughput of TCP New Reno and Vegas over congested link

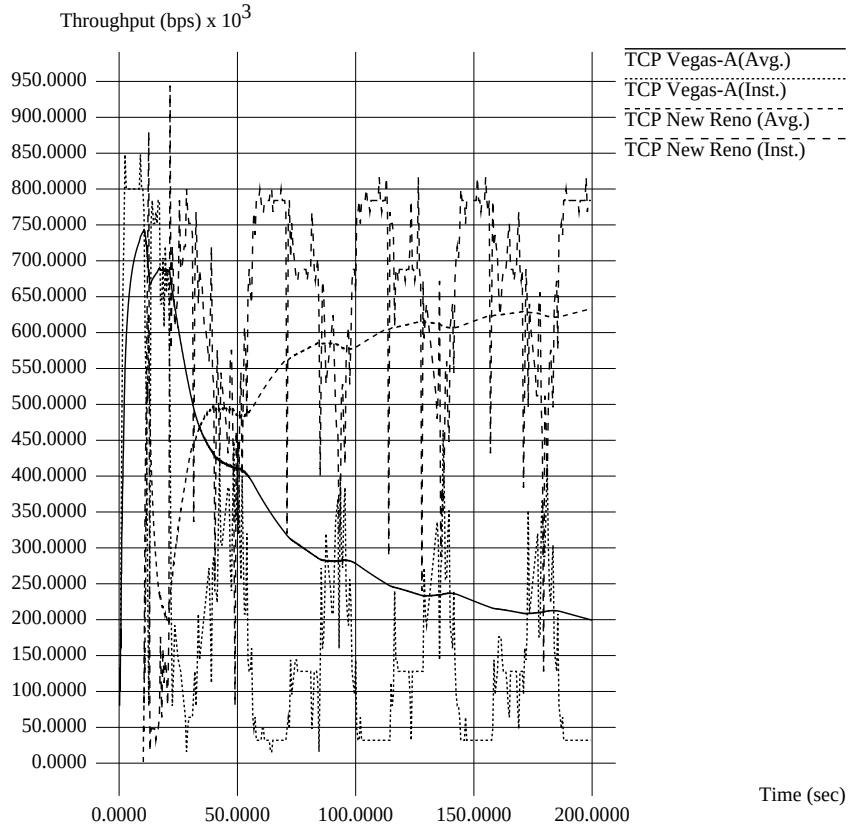

Fig.16. Throughput of TCP New Reno and Vegas-A connections over congested link

When the above experiment was repeated with 3 New Reno sources and 3 Vegas/Vegas-A sources, TCP Vegas-A was found getting a more than fair share of the congested link bandwidth. Each source-to-R1 link had a bandwidth of 1Mbps and RTT of 20ms. R1-R2 link had a bandwidth of 1Mbps and RTT of 80ms. R2-to-destination link was 1Mbps bandwidth limited and had RTT of 20ms.

Figures 17 and 18 show the average throughputs obtained. The Vegas/Vegas-A sources (S1, S2, S3) were started at 0,10,20 seconds while the New Reno sources (S4, S5, S6) were started at 30,40,50 seconds. The average throughput obtained for each flow is given in Table 6.

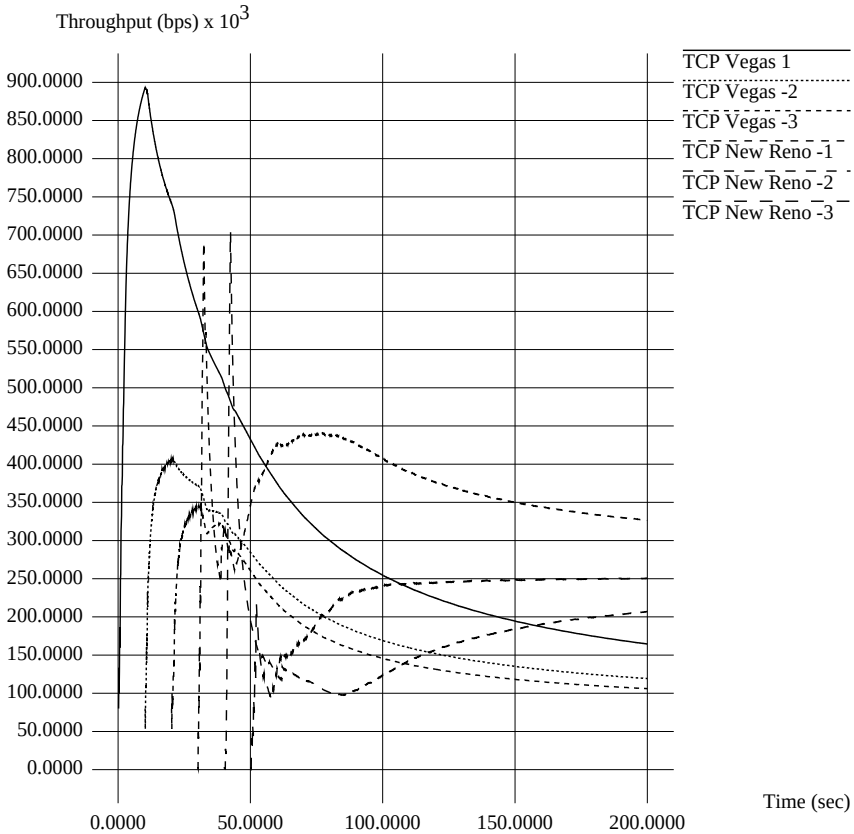

Fig. 17. Throughput of 3 TCP New Reno and 3 Vegas connections over congested link

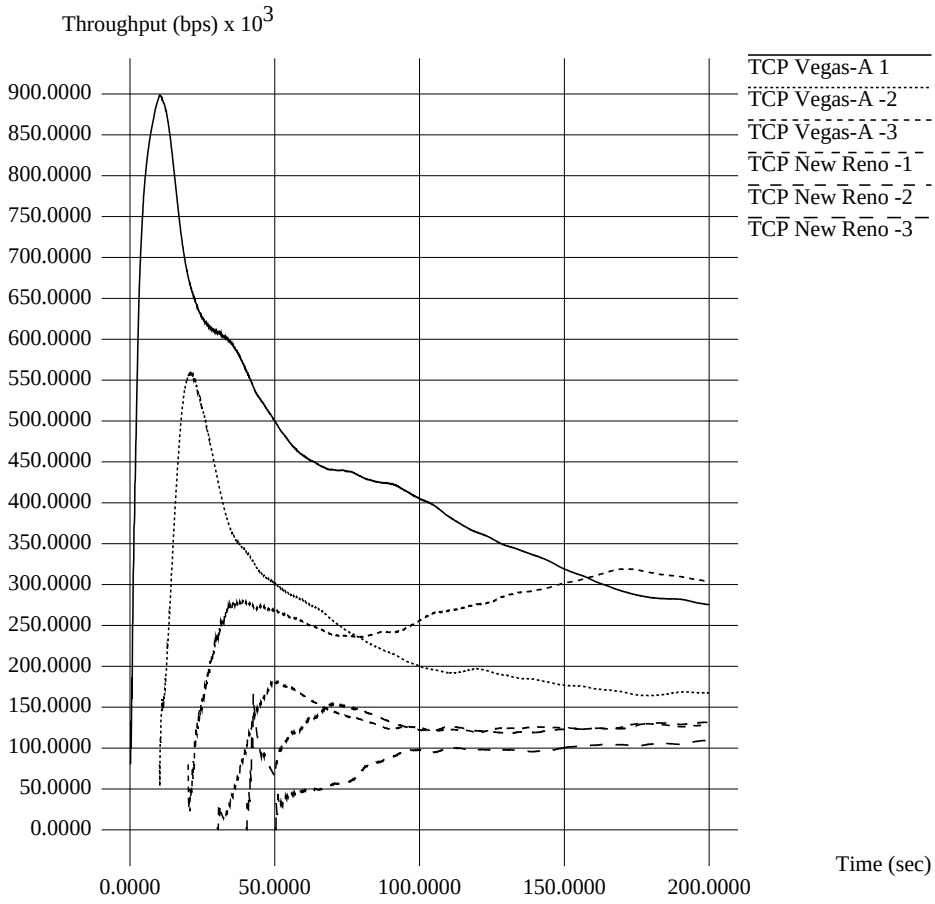

Fig. 18. Throughput of 3 TCP New Reno and 3 Vegas-A connections over congested link

Sources	Vegas & New Reno	Vegas-A and New Reno
S1	164480	275480
S2	119242	167789
S3	106044	304178
S4	326590	127908
S5	207002	131402
S6	250828	109336

Table 6. Throughputs of New Reno-Vegas and New Reno-Vegas-A connections

From the values in Table 6, the "interprotocol fairness ratio" [31] of the Vegas and New Reno connections can be calculated as 0.497 and that of the Vegas-A and New Reno connections as 2.028. Interprotocol fairness ratio is defined as the ratio of the average bandwidth shared between the two protocols, i.e., it is the ratio of average TCP Vegas/Vegas-A bandwidth calculated across all the Vegas/Vegas-A flows to the

average TCP New Reno bandwidth calculated across all the New Reno flows. The obtained values show that Vegas-A actually outperforms New Reno.

C. Fairness between old and new connections

As mentioned in section 2.2.1, TCP Vegas has the disadvantage that newer connection of Vegas enjoys a larger throughput because of the discrepancy in the estimation of baseRTT. Vegas-A's modified congestion control mechanism overcomes this shortcoming. To illustrate this, a scenario was created where 5 TCP Vegas/Vegas-A connections share a single link. The source-to-router links were set to 1Mbps and RTT to 20ms, while the router-to-destination links' bandwidth was set to 1Mbps and RTT to 100ms. The sources were started at intervals of 50 seconds each.

Table 7 shows the average throughputs obtained by the connections for the simulation lasting 900 seconds. Figure 19 shows the average throughputs (in bytes/sec) of the Vegas connections, while Figure 20 shows that of the Vegas-A connections.

Sources	Vegas	Vegas-A
S1	218531	221447
S2	191533	199760
S3	206176	247431
S4	247585	229577
S5	266913	234662
Std. Deviation	33217.1	17711.1

Table 7. Throughput of five Vegas and Vegas-A connections

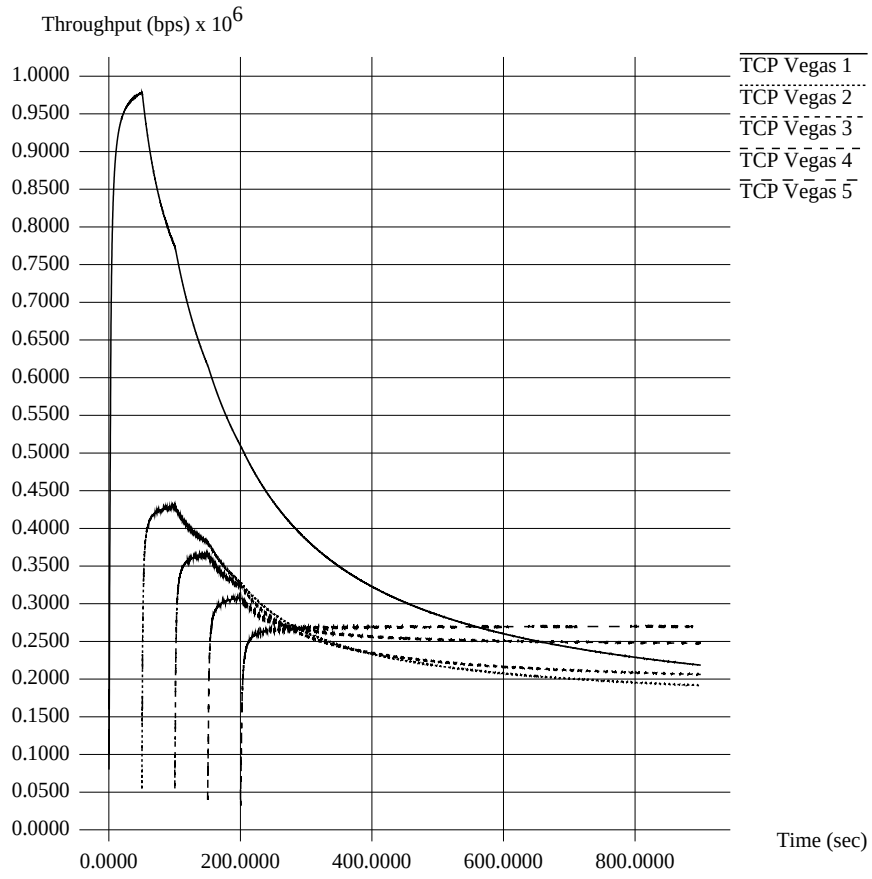

Fig. 19. Throughput of 5 Vegas connections over congested link

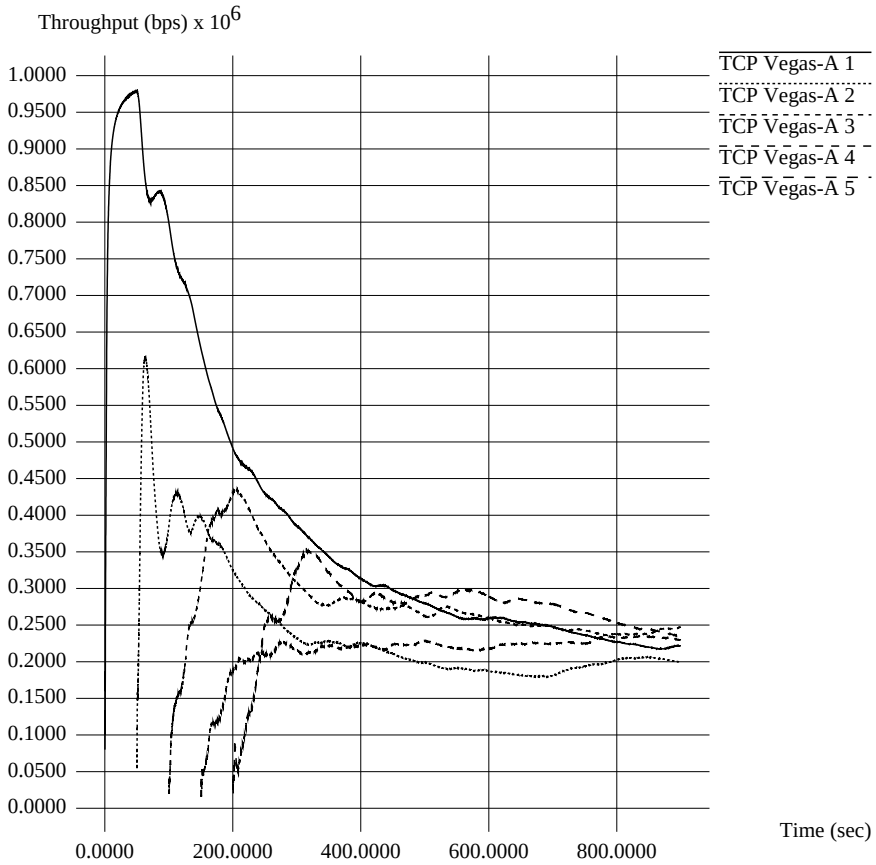

Fig. 20. Throughput of 5 Vegas-A connections over congested link

It is obvious from Figure 19 that new connections enjoy larger throughputs compared to older connections in the case of Vegas. On the other hand, with Vegas-A, this problem is somewhat reduced, as seen in Figure 20. It can be seen from Table 7 that, the standard deviation of the average throughput of Vegas-A connections is less than that of Vegas connections. This means that Vegas-A connections share the bandwidth in a fairer manner compared to the Vegas connections.

D. Bias Against high bandwidth flows

Hasegawa et al. [21] show that TCP Vegas, like Reno, has the fairness bias against connections with higher bandwidth. To test whether Vegas-A can perform better than Vegas in terms of fairness, simulations were conducted with 3 connections S1-D1, S2-D2, and S3-D3 and with the S1-R1 link bandwidth limited to 128Kbps, S2-R1 to 256Kbps, and S3-R1 to 512Kbps. The RTT of each source-to-R1 link was set to 5ms. R1-R2 link was bandwidth limited to 400Kbps and the RTT was set to 10ms. R2-to-destination link had a bandwidth of 1Mbps and RTT of 5ms. The simulations were carried out for a period of 1800 seconds. Table 8 summarises the throughputs (in Kbps) obtained and compares it with the expected values. With the Si-R1 link bandwidth represented by bw_i, the expected value for connection Si-Di is calculated as $bw_i * R / \sum_1^3 bw_i$ where R is the bottleneck link bandwidth.

	S1	S2	S3
Expected	57.14	114.29	228.57
Vegas	123.34	146.85	120.46
Vegas-A	98.90	134.54	158.25

Table 8. Comparison of Vegas and Vegas-A bias against high bandwidth connection for 1800 seconds

Ideally, S3-D3 connection should have received 228Kbps of the bottleneck bandwidth, but when Vegas is used, the connection ends up with just 120.5Kbps bandwidth. However, when the source is switched to Vegas-A, the three connections enjoy fairer shares of the bandwidth (closer to the expected values). In the case of Vegas, the *cwnd* attains a steady value, such that $\alpha<diff<\beta$, in the very early stage of the connection, and remains like that for the entire duration of the connection. In the case of Vegas-A, α and β can vary dynamically, allowing *cwnd* to probe the network for more available bandwidth.

E. Retaining properties of Vegas

TCP Vegas-A tries to improve upon the congestion control mechanism of TCP Vegas by trying to dynamically adapt to the availability of network bandwidth. However, it is essential that in doing so, Vegas-A should not lose the properties of Vegas that set it apart from New Reno. To ensure that Vegas-A does inherit the properties of Vegas, simulations were conducted with only one Vegas/Vegas-A/New Reno source connected through the routers to the destination. The S1-R1 link bandwidth was set to 1Mbps and the RTT of the link to 5ms and 45ms, respectively, for two sets of experiments. The R1-R2 bandwidth was restricted to 250Kbps and the RTT to 5ms and 45ms, respectively, for the two sets of experiments. The R2-D2 link bandwidth was set to 1Mbps and RTT to 10ms. File sizes of 10MB and 5MB were sent from source to the destination. Tables 9 and 10 show the values recorded for the time taken for the complete transfer, the average queue length at the router (in packets), and the number of retransmitted packets.

Source	5MB file			10MB file		
	Time	Avg. Queue	Retx. Pkts.	Time	Avg. Queue	Retx. Pkts.
New Reno	162.353	11.46	58	322.353	23.27	62
Vegas	160.253	0.65	0	320.253	1.3	0
Vegas-A	160.253	5.62	0	320.256	12.4	0

Table 9. Comparison of New Reno, Vegas and Vegas-A connections over a 20ms RTT link

Source	5MB file			10MB file		
	Time	Avg. Queue	Retx. Pkts.	Time	Avg. Queue	Retx. Pkts.
New Reno	166.505	11.29	59	326.505	23.0	62
Vegas	160.651	0.82	0	320.651	1.63	0
Vegas-A	160.654	4.85	0	320.651	10.84	0

Table 10. Comparison of New Reno, Vegas and Vegas-A connections over a 100ms RTT link

As the tables show, Vegas and Vegas-A outperform New Reno in all performance measures, as expected. The only difference between Vegas and Vegas-A is that the average buffer occupancy at the routers is larger for Vegas-A compared to Vegas. This is expected since Vegas-A adapts α and β to values larger than 1 and 3, which in turn lets Vegas-A increase the congestion window to a larger value and thus pushes in more data into the network. However, note that this larger average buffer occupancy does not seem to increase the time required to complete the transfer of the files, thus showing that the queuing delay has not increased much. It also does not fill the router queue completely. If it had, packets would have been dropped and hence retransmitted.

Next, the performance of Vegas, Vegas-A and New Reno were studied as the router queue sizes were varied. The RTT of the source-to-destination links were fixed at 40ms. S1-R1 and R2-D1 link bandwidths were set at 1Mbps, while the R1-R2 link bandwidth was set at 500Kbps. Files of size 10MB was sent from S1 to D1. Table 11 shows the values obtained. The unit of time is seconds and the retransmissions record the number of packets retransmitted.

Buffer size →	10		15		20		25		30	
Source	Time	Retx.	Time	Retx.	Time	Retx.	Time	Retx.	Time	Retx.
New Reno	161.3	106	161.6	74	161.9	61	162.2	56	162.5	55
Vegas	160.4	0	160.4	0	160.4	0	160.4	0	160.4	0
Vegas-A	160.5	2	160.6	1	160.6	1	160.4	0	160.4	0

Table 11. Comparison of New Reno, Vegas and Vegas-A connections with different router buffer queue size

The results show that when the buffer size is as small as 10,15 or 20 packets, Vegas-A has to retransmit at most 1 or 2 packets, while Vegas does not lose any packets at all. This number is extremely small compared to the number of packets dropped and retransmitted when New Reno is used. Furthermore, when the queue size is increased to 25 and 30, the behaviour of Vegas-A approaches that of Vegas. Under normal circumstances, a router usually has buffer sizes greater than 20 packets.

4.2.2 Vegas-A in satellite environment

There has not been any previous investigation on the performance of TCP Vegas over satellite links. The peculiar characteristics of satellite links warrant an investigation on how TCP Vegas and Vegas-A perform, when compared to TCP New Reno in these environments. Satellite systems can broadly be categorised into GEO and LEO satellites. In this section we simulate the satellite network of both GEO and LEO systems using the NS extensions.

(A) GEO Satellite

In this set of simulations, Vegas-A's performance over a GEO satellite link between New York and San Francisco was analysed. The uplink and downlink bandwidth was set to 1.5Mbps. The terminals were set to use 'DropTail' queue of 50 packets. FTP traffic was generated using various TCP mechanisms like New Reno, Vegas and Vegas-A, from New York to San Francisco. Various scenarios were

simulated, with varying packet error rates (PER), single source traffic and competing traffics. A packet error rate of 0.01 means that 1 in 100 packets are corrupted. For each value of PER, the simulation was performed five times using different random seed values for error generation and the average of the results obtained were noted for the following sub-sections.

1. One source, varying packet error rate

In this set of simulations, one FTP source was run while the PER of the satellite links were varied. The simulation was run for 600 seconds to give the connection time to tide over transient conditions. Table 12 shows the results obtained when various TCP versions were used as the FTP source. The throughputs are in bits/sec and number of retransmission in units of packets.

PER	New Reno		Vegas		Vegas-A	
	Throughput	Retrx.	Throughput	Retrx.	Throughput	Retrx.
0	1208387	163	1444300	0	1444474	0
0.0005	675312	189	1097909	53	1098457	52
0.005	263907	112	407709	149	407759	149
0.05	64828	239	103131	371	102557	371

Table 12: Throughput and number of retransmissions for New Reno, Vegas and Vegas-A connections using GEO satellite links

As we can see from the table above, Vegas and Vegas-A performs almost identically and their throughputs are higher than that of TCP New Reno's. Note that the higher amount of packet drops in case of PER>0 for Vegas and Vegas-A is because of the fact that since throughput is higher, more packets are pumped into the network and hence more packets are corrupted. Vegas and Vegas-A do not lose packets due to congestion, as the values for PER of 0.0 cases will show.

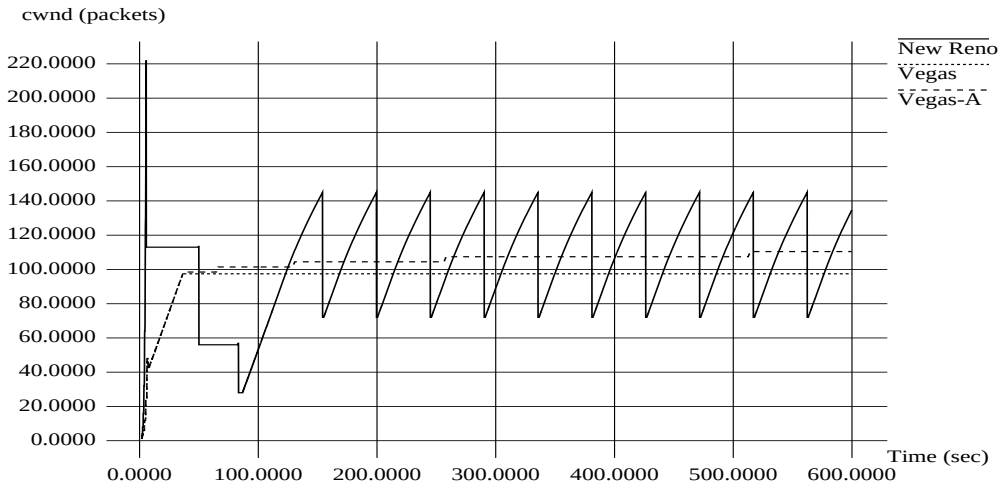

Fig. 21. *cwnd* variation for New Reno, Vegas and Vegas-A connections when PER=0.0

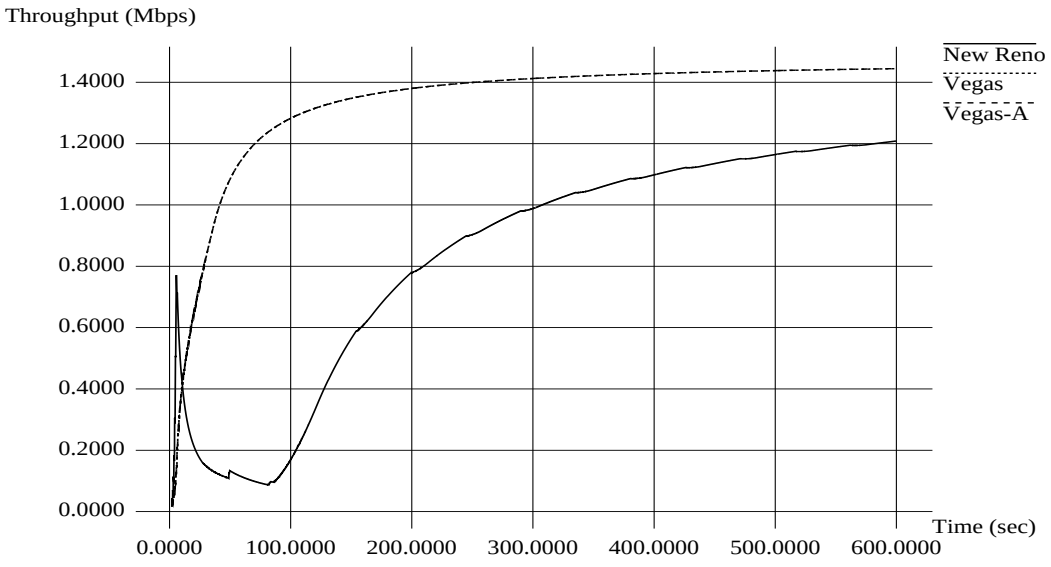

Fig. 22. Throughput of New Reno, Vegas and Vegas-A connections when PER=0.0

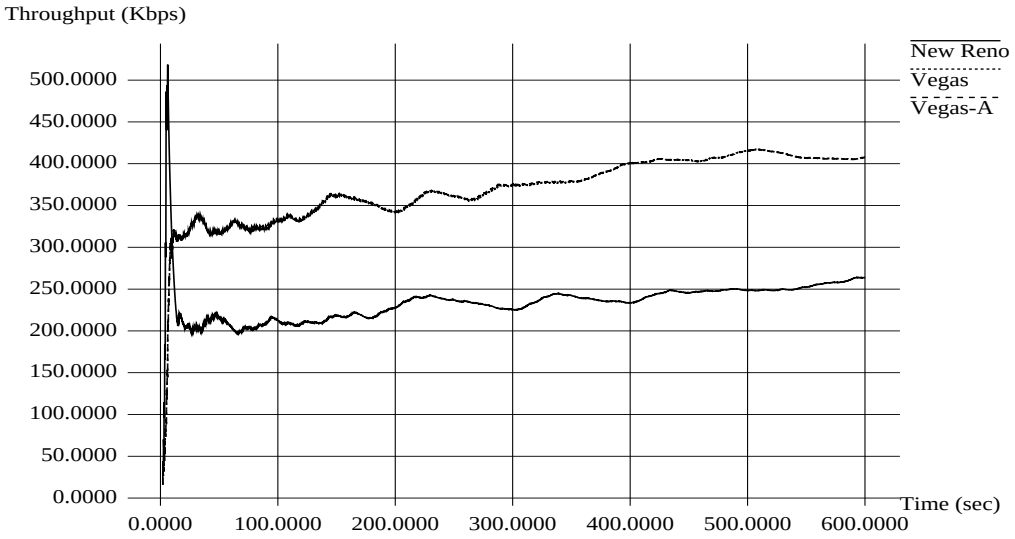

Fig. 23. Throughput of New Reno, Vegas and Vegas-A connections when PER=0.005

When PER is 0.0 (Figure 21), the *cwnd* of New Reno shows the typical saw tooth format, while Vegas's *cwnd* remains constant at around 100 packets. However, just like in the case of the wired network, Vegas-A's *cwnd* keeps on increasing slowly but steadily without any fall. As a result, Vegas and Vegas-A perform better than New Reno in terms of throughput (Figure 22) and number of retransmissions. As Vegas and Vegas-A perform identically, their graphs coincide. As can be seen from Figure 23, when PER=0.005 the frequent packet loss reduces the New Reno throughput to around 260Kbps, while for Vegas and Vegas-A, the throughput climbs all the way to 400Kbps. Again, the performance of Vegas and Vegas-A are almost the same and hence, their graphs coincide.

2. Competing connections, varying PERs

In this set of simulations, effect of presence of competing New Reno traffic on the performance of the TCP connections was studied. A New Reno/Vegas/Vegas-A connection was started in the beginning and after 10.0 seconds, another New Reno traffic was started. Both the sources were attached to FTP applications. The simulation was run for 600 seconds to make sure that the readings obtained were at stable conditions. Tables 13 to 16 below details the various throughputs obtained for various TCP source combinations and PERs.

		Throughput (bps)	Goodput (bps)	Lost (packets)
New Reno Vs New Reno	New Reno	628340	626017	174
	New Reno	611932	609559	175
Vegas Vs New Reno	Vegas	368520	368507	1
	New Reno	1069722	1967783	143
Vegas-A Vs New Reno	Vegas-A	487252	487131	9
	New Reno	936176	934210	145

Table 13. Throughput and goodput at PER 0.0

		Throughput (bps)	Goodput (bps)	Lost (packets)
New Reno Vs New Reno	New Reno	526491	523993	187
	New Reno	660461	658807	122
Vegas Vs New Reno	Vegas	552440	552146	22
	New Reno	734386	732285	155
Vegas-A Vs New Reno	Vegas-A	592854	592520	25
	New Reno	724678	722997	124

Table 14. Throughput and goodput at PER 0.0005

		Throughput (bps)	Goodput (bps)	Lost (packets)
New Reno Vs New Reno	New Reno	282698	281336	102
	New Reno	249031	246929	155
Vegas Vs New Reno	Vegas	380353	378390	147
	New Reno	269464	268217	92
Vegas-A Vs New Reno	Vegas-A	411886	409989	142
	New Reno	242332	240841	110

Table 15. Throughput and goodput at PER 0.005

		Throughput (bps)	Goodput (bps)	Lost (packets)
New Reno Vs New Reno	New Reno	62224	58885	250
	New Reno	67824	64597	238
Vegas Vs New Reno	Vegas	93515	88534	373
	New Reno	64976	61776	236
Vegas-A Vs New Reno	Vegas-A	94797	89709	381
	New Reno	64881	41410	256

Table 16. Throughput and goodput at PER 0.05

In Tables 13 and 14, when PER is very small, the competing New Reno flow had a more than fair share of the bandwidth. Vegas flow lost out to New Reno flow. However, Vegas-A flow was able to compete better and achieve better throughput compared to Vegas. When PER was increased, (Tables 15 & 16), the competing New Reno flow's throughput decreased to a value lesser than that of the competing Vegas or Vegas-A connection. Again, Vegas-A connection is able to achieve a better throughout compared to Vegas.

The Figures 24, 26, 27 and 28 show the average throughout of the connections at various PERs, while Figure 25 shows the cwnd variation of the various connections when the PER is 0.0. As can be seen from Figure 24, Vegas and Vegas-A's average throughput decreases when the competing traffic starts and goes down all the way

below 500Kbps at the end of 600 seconds. However, Vegas-A's throughput is always better than that of Vegas.

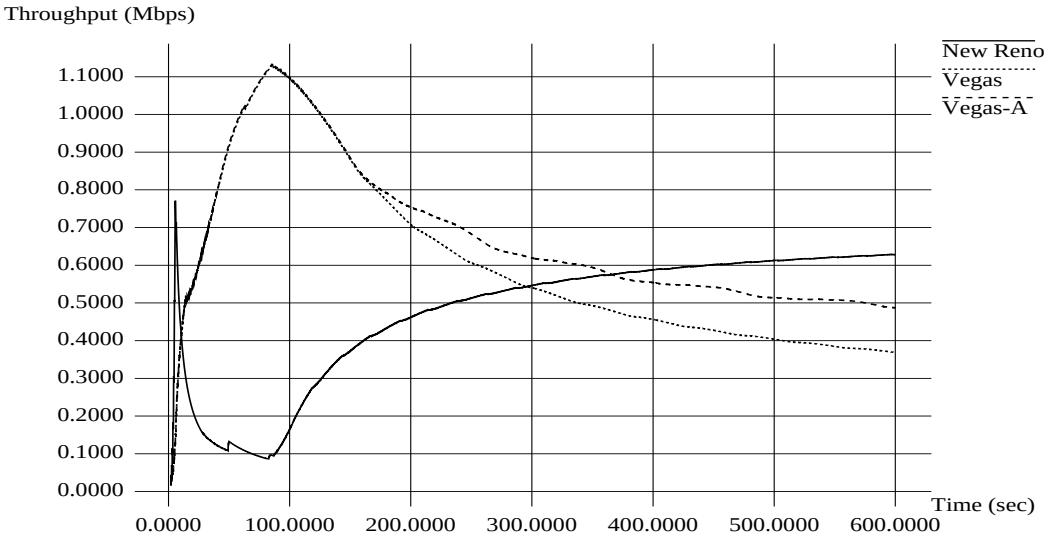

Fig. 24. Average throughput for New Reno, Vegas and Vegas-A connections when competing with another New Reno connection with PER=0.0

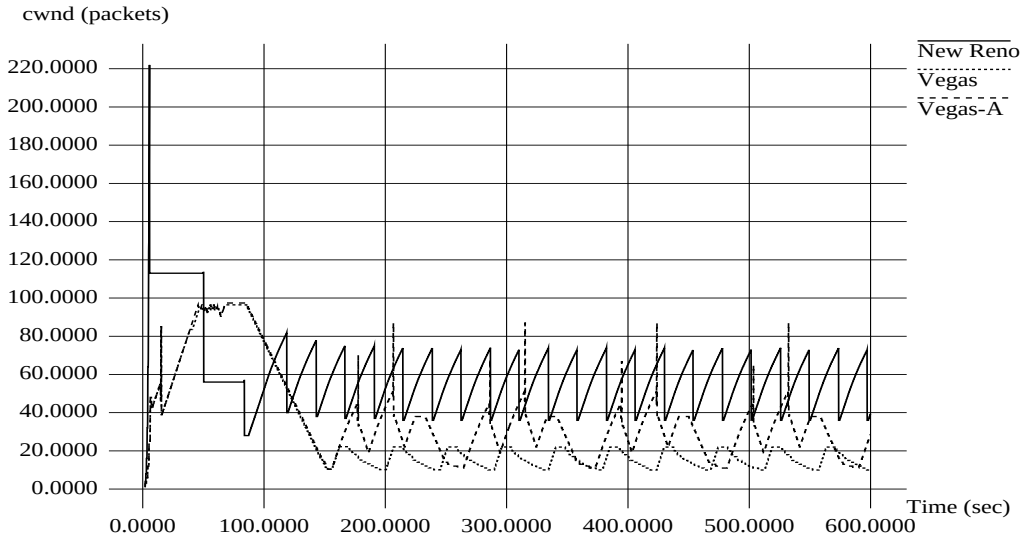

Fig. 25. *cwnd* variation for New Reno, Vegas and Vegas-A connections when competing with another New Reno connection with PER=0.0

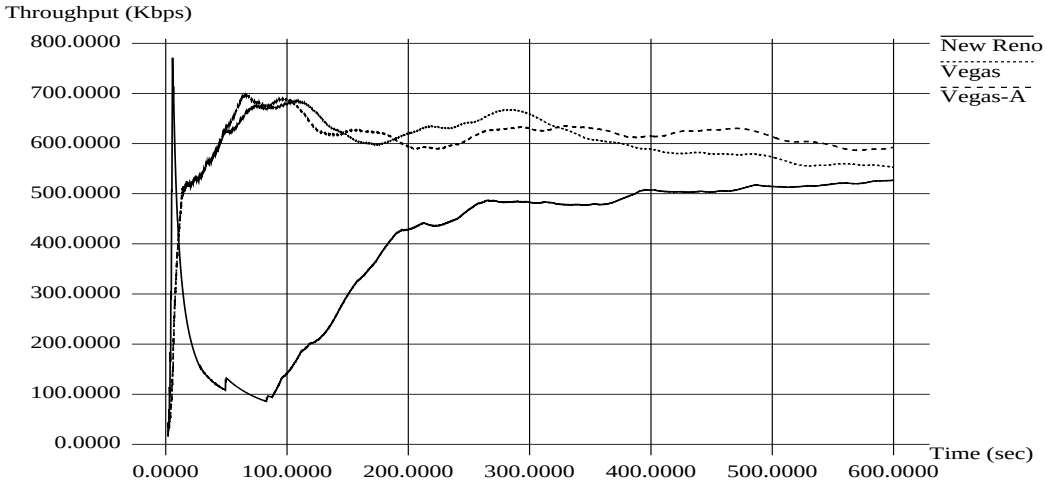

Fig. 26. Average throughput for New Reno, Vegas and Vegas-A connections when competing with another New Reno connection with PER=0.0005

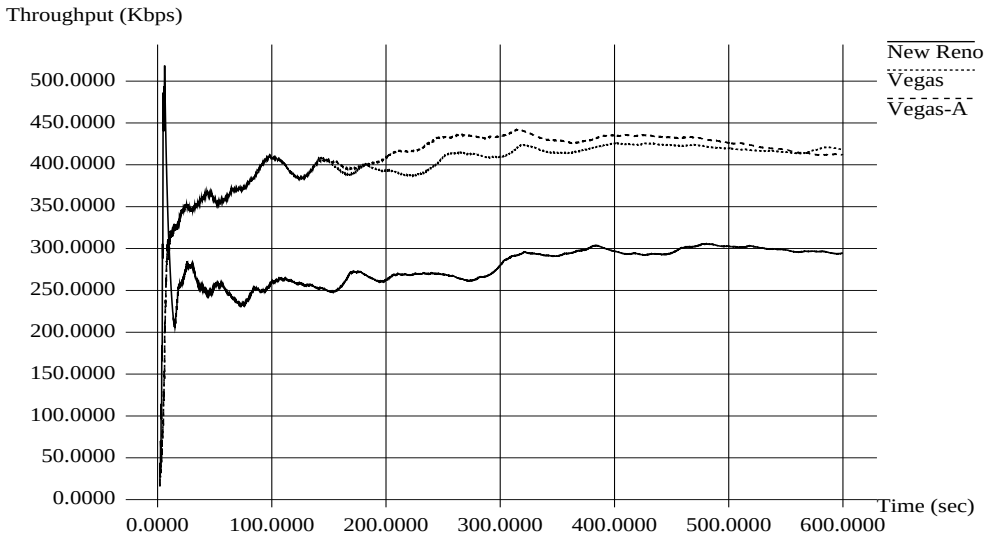

Fig. 27. Average throughput for New Reno, Vegas and Vegas-A connections when competing with another New Reno connection with PER=0.005

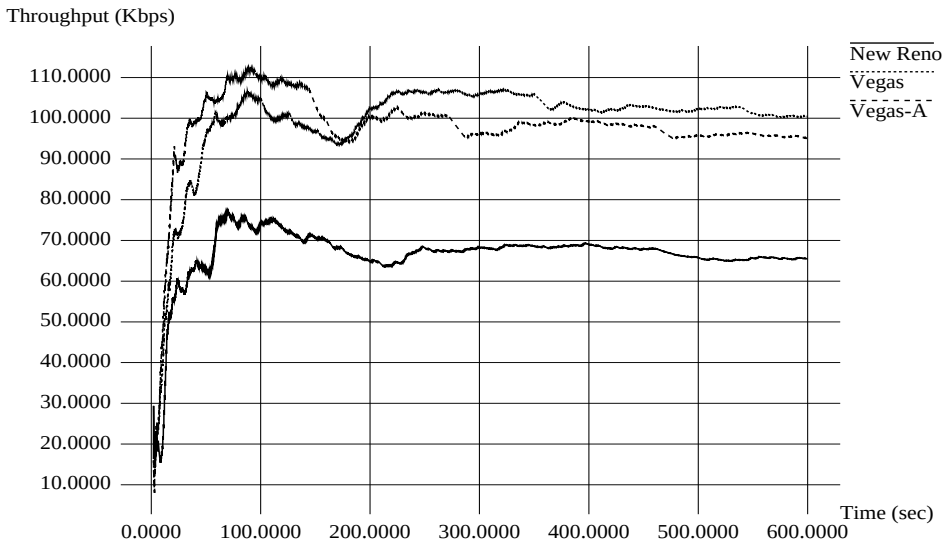

Fig. 28. Average throughput for New Reno, Vegas and Vegas-A connections when competing with another New Reno connection with PER=0.05

Figure 25 explains the reason for the better throughput of Vegas-A seen in Figure 24. While Vegas's *cwnd* stays below 20 packets when New Reno traffic competes with it, Vegas-A is able to increase its *cwnd* to an average of 30 packets. Figures 26, 27 and 28 shows that when PER increases, the throughout of Vegas and Vegas-A increases beyond that of the New Reno connection. This reason for this behaviour can be seen from Figures 29 and 30. It can be seen that from 100 seconds to 400 seconds, New Reno goes to slow start at least 25 times, while Vegas and Vegas-A performs slow start only 2-3 times during the same period. This is due to the early segment loss detection algorithm used by Vegas, which prevents slow start from kicking in.

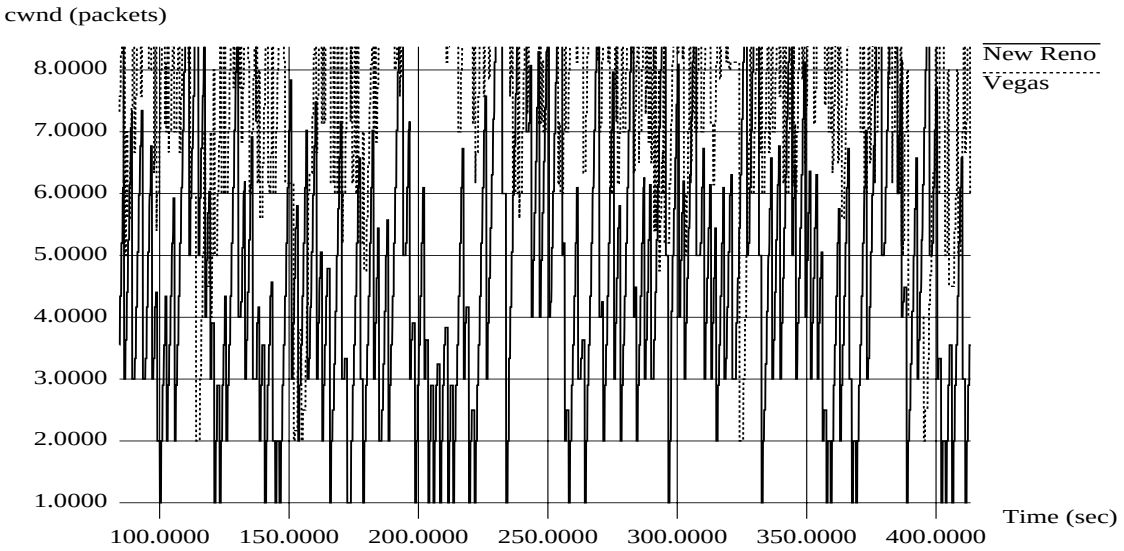

Fig. 29. *cwnd* variation for competing New Reno and Vegas connections with PER=0.05

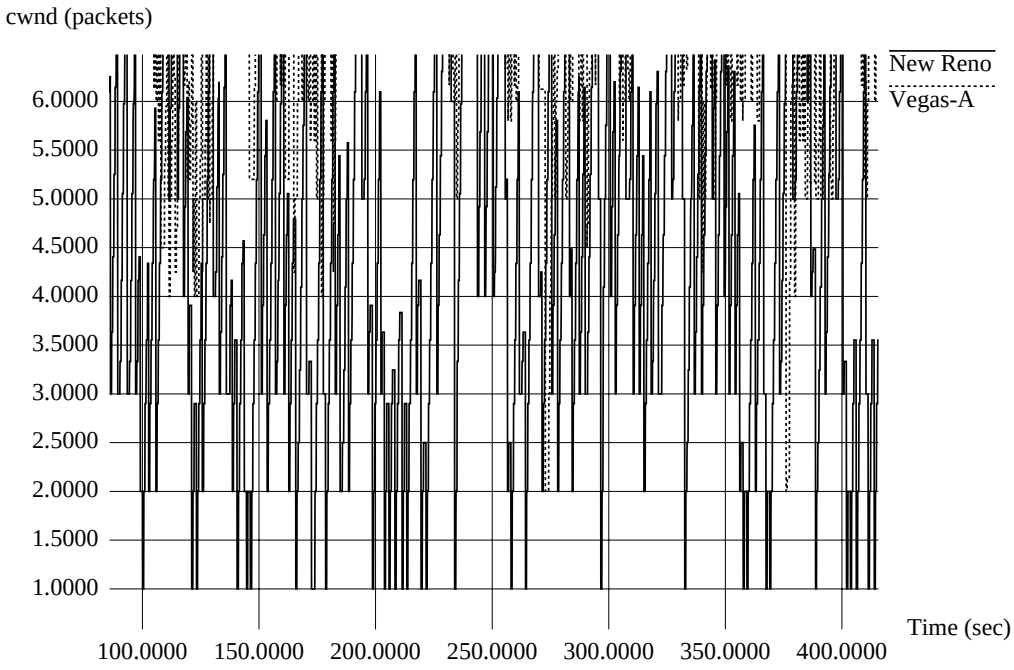

Fig. 30. *cwnd* variation for competing New Reno and Vegas-A connections with PER=0.05

(B) Low Earth Orbit (LEO) Satellite

Next, LEO satellite environment was simulated, again using NS2. For this modelling, the Iridium satellite system was chosen, whose altitude is 780 Km above earth. The satellites have an orbital period of 6206.9 seconds, with inter-satellite separation of 32.72 degrees and seam separation of 22 degrees. The two connection terminals were simulated at Berkeley and Boston. All simulations were run for 600 seconds.

1. Single source, 0.0 PER

A single source of TCP connection was simulated to carry FTP data over LEO satellite links with no errors in the link. New Reno, Vegas and Vegas-A traffic was simulated and throughput attained was recorded.

	Throughput (bps)	Lost packets
Vegas	1348086	0
Vegas-A	1459432	52
New Reno	1455693	189

Table 17. Throughput of various TCP connections over LEO satellite with 0.0 PER

As can be seen from Table 17 and Figure 31, Vegas-A's throughput is the largest, while Vegas's is the lowest. It can be seen that until about 375s, Vegas's and Vegas-A's performance are comparable. However, from 375s to around 525s, Vegas's throughput decreases considerably. It begins to pick up slowly again after 525ms, while the performance of Vegas-A and New Reno remains almost steady.

This observation can be explained using Figures 32 and 33. Figure 32 shows that the RTT of the connections started to increase slowly at around 375 seconds. This increase could be due to the handover mechanism of the LEO satellite system. This caused the *cwnd* (Figure 33) of Vegas-A and New Reno to increase to around 65 packets, thus allowing it to better utilize the larger BW*D product. However, Vegas assumes that this increase in RTT is due to congestion and hence starts to decrease the *cwnd* to around 12 packets, thus not utilizing the larger BW*D product. This goes on until 525s, when RTT decreases again, forcing New Reno and Vegas-A to decrease the *cwnd* again. This better utilization of BW*D product between 375 sec and 525 seconds by Vegas-A allows it to outperform Vegas.

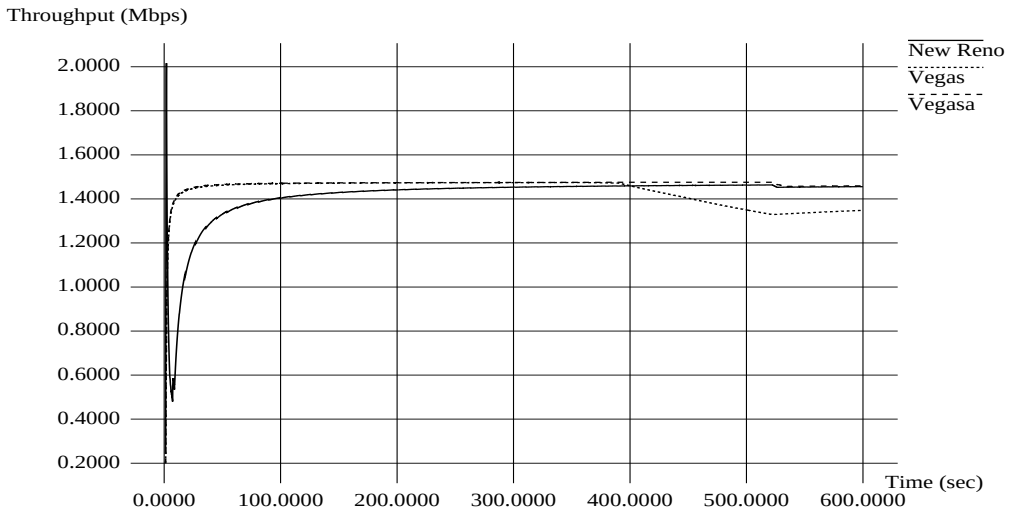

Fig. 31. Average throughput of New Reno, Vegas and Vegas-A connections over LEO links when PER=0.0

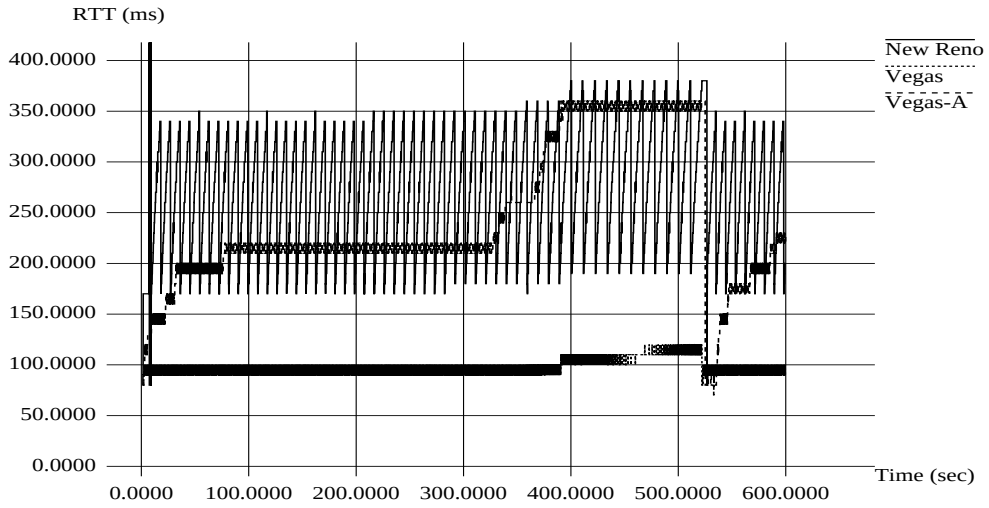

Fig. 32. RTT variation of New Reno, Vegas and Vegas-A connections over LEO links when PER=0.0

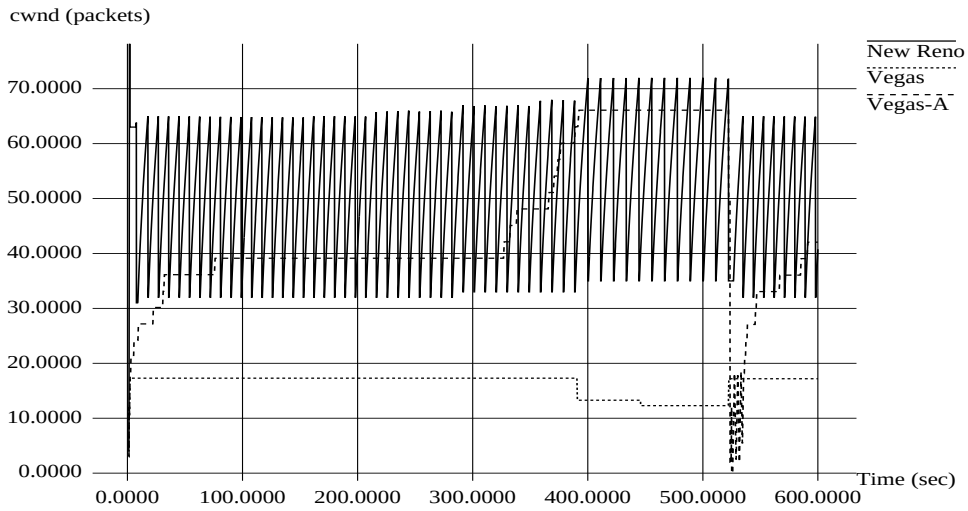

Fig. 33. cwnd variation of New Reno, Vegas and Vegas-A connections over LEO satellite links when PER= 0.0

2. Competing traffic, PER 0.0

Next, competing traffic scenario was simulated to see how Vegas and Vegas-A fares when it competes with a New Reno connection for the available bandwidth. The competing New Reno connection starts 10 seconds after the first source's connection was established. It was found that changing the order of connections yielded similar results.

		Throughput (bps)	Goodput (bps)	Lost (packets)	% share of goodput
New Reno Vs New Reno	New Reno	742878	740114	207	47.30
	New Reno	734509	732610	140	52.7
Vegas Vs New Reno	Vegas	154484	154417	50	9.80
	New Reno	1340244	1338156	154	90.20
Vegas-A Vs New Reno	Vegas-A	387005	386337	50	24.2
	New Reno	1104719	1102712	148	75.8

Table 18. Throughput of various TCP connections when competing with TCP New Reno source over LEO satellite with 0.0 PER

As evident from the numbers in the table above, Vegas performs disastrously when it has to compete with New Reno connection, getting as low as 10% of the total bandwidth. However, Vegas-A is able to compete better and gets as much as 24% of the bandwidth. Figures 34 to 37 show the fluctuation of the average throughput of the various sources over a period of time.

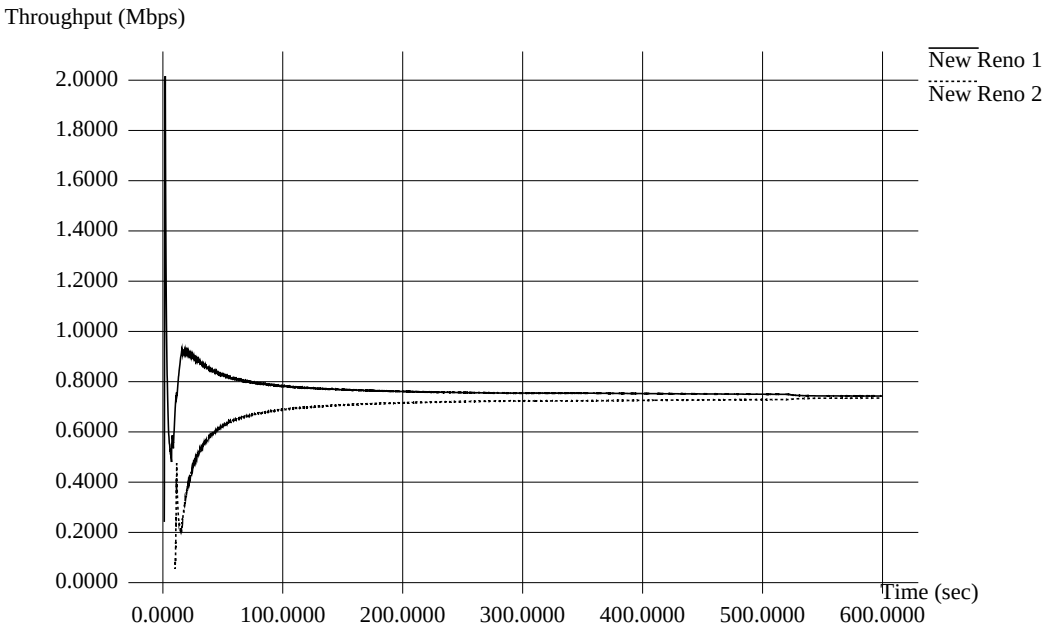

Fig. 34. Average throughput variation of two competing New Reno traffic over LEO satellite links when PER=0.0

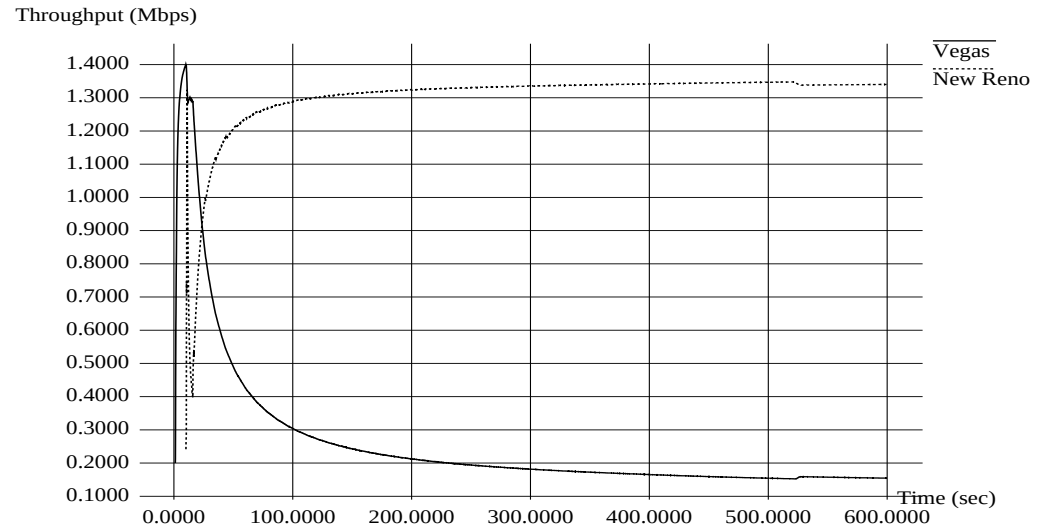

Fig. 35. Average throughput variation of competing New Reno and Vegas traffic over LEO satellite links when PER=0.0

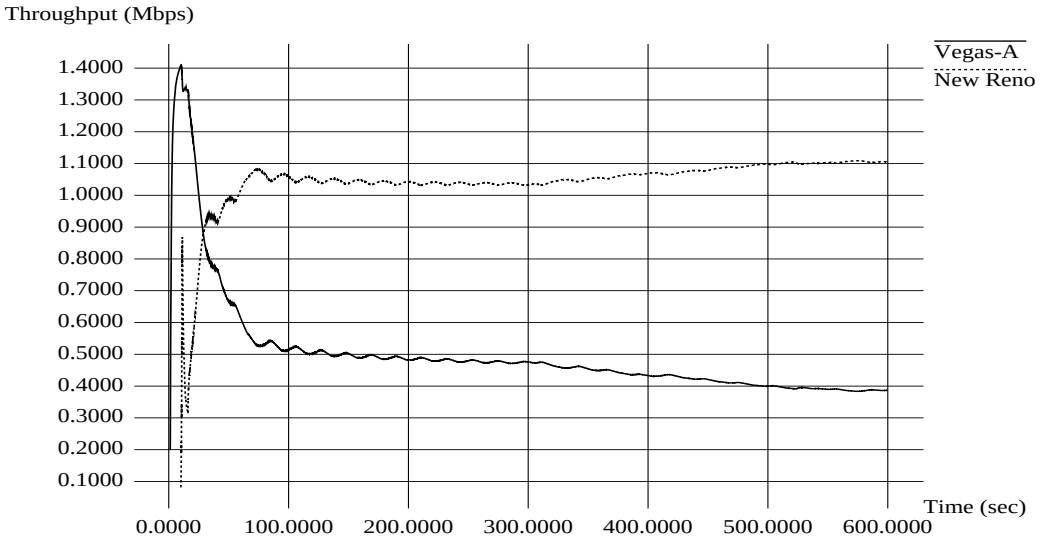

Fig. 36. Average throughput variation of competing New Reno and Vegas-A traffic over LEO satellite links when PER=0.0

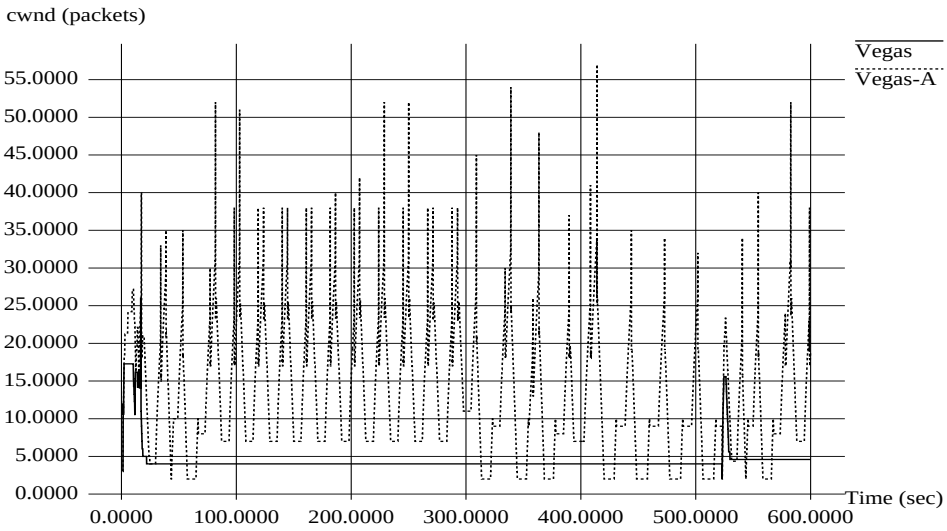

Fig. 37. Variation of Vegas's and Vegas-A's *cwnd* when competing with another New Reno flow over LEO satellite links when PER=0.0

4.3 Summary

In this chapter, a modified version of TCP Vegas, named TCP Vegas-A was proposed to mitigate some of the limitations of TCP Vegas. To test the performance of the modified algorithm, simulations were conducted on both wired and satellite environments.

It was shown that in a wired environment, Vegas-A performs better than Vegas when competing with other TCP connections like New Reno for shared bandwidth. Vegas-A is able to overcome the re-routing limitations of Vegas and is able to adapt to the changes in RTT and routes faster. It was also shown that Vegas-A connections do not suffer from unfairness towards old connection and unfairness against higher

bandwidth connections problems of Vegas. It was shown that even though Vegas-A is different from Vegas, the basic congestion control algorithm still remains as effective as Vegas in decreasing the average queue occupancy and the number of packets retransmitted by the connection.

In a satellite environment, it was shown that TCP Vegas-A performs better than Vegas. It was shown that Vegas-A is able to compete against New Reno more successfully than Vegas, without losing out to Vegas in any other conditions. Simulations proved that Vegas-A was able to handle the varying RTT behaviour of LEO satellite systems, while Vegas reacted badly to such fluctuations.

Chapter 5

WORST-CASE PERFORMANCE LIMITATION OF TCP SACK AND A FEASIBLE SOLUTION

In this chapter one of TCP SACK's worst-case limitation is looked into and a proposal is put forward to modify the present SACK's implementation to overcome this limitation. Simulated results are presented to support the proposal.

5.1 Limitation of TCP SACK

The limitation of TCP SACK is mentioned in detail in section 2.2.2. It is being condensed and mentioned here for completeness. TCP SACK requires 64 bits (8 bytes) to represent the upper and lower bound sequence numbers of every block it selectively acknowledges. As the TCP protocol limits the maximum length of the options field to 40 bytes and SACK is usually implemented along with TCP Timestamp options, an acknowledgment packet can carry a maximum of only three blocks' information. This rather small maximum number of SACK block information can lead to efficiency problems in the performance of TCP. Under certain packet loss scenarios, the TCP sender will end up retransmitting packets that have already been received successfully.

5.2 Alternate Proposal

To overcome the above-mentioned worst-case limitation of TCP SACK, the following is proposed. The packet structure of TCP cannot be changed. Hence the aim should be to reduce the amount of data needed to represent a SACK block. The proposal is to modify the SACK option format shown in Figure 2, as follows:

- Instead of sending the absolute 32-bit sequence number for all the edges of all the blocks, send 32-bit sequence number for only the right edge of the 1st block (let us denote it by A). For the rest of the edges, we consider two different alternative means of representation:

  - In the first alternative, send the offsets measured from the edge A. We denote them by O_{12}, O_{21}, O_{22}… O_{n1}, O_{n2}, where O_{12} is the offset of the left edge of first block from A, O_{21} & O_{22} are respectively the right and left edges of the second block, and so on.
  - The second alternative is to represent each edge as an offset from the previous edge. So for the left edge of the first block (O_{12}), compute the offset from A. For O_{21} compute it with respect to O_{12}. For O_{22} compute it with respect to O_{21}, and so on.

- Find out the biggest number among these offsets (denote it by O_{max}). Let X be $\lceil \log_2 (O_{max}) \rceil$ (where $\lceil x \rceil$ is the smallest integer larger than x). This means that we can represent all the offsets using 'X' bits. We need to send this number 'X' too to the data sender within the SACK option fields. Thus the proposal is to change the format of the SACK as in Figure 38.

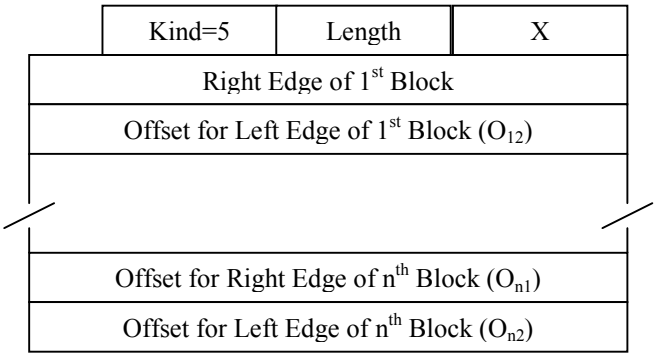

Fig. 38. Proposed SACK option format

For the rest of the paper, the first alternative is used. From the proposed format shown in Figure 38 note that:

- As the sequence numbers range from 0 to 2^{32}-1, the maximum value that X can take is 32. 5 bits are needed to send the value of X. To keep it simple, one byte is allocated for this purpose. This is the extra byte that the new format has after the 'Length' field, labeled as 'X'.
- The first field after 'X' will be the right edge of the 1st block - a 32-bit sequence number.
- The next field (O_{12}) is the offset of the left edge of the 1st block with respect to the right edge. This number can be represented using X bits (not the usual 32 bits!).
- All the offsets are computed with respect to the right edge of the 1st block, as this is the only absolute 32-bit sequence number that will be sent to the data sender.

At the TCP sender side, when it receives an ACK with SACK data in it, it reads the value of X and A. Then, using blocks of X bits, it reads all the offset values and recreates the edges by adding the offset to the value of A.

5.3 Example

It is easier to understand the proposed changes using an example. The following is one of the examples used in RFC 2018 [10]. It is used here so that the difference can be compared. Assume the starting sequence number is 5000 and that the data transmitter sends a burst of 8 segments, each containing 500 data bytes.

In the scenario presented, every other segment (2nd, 4th, 6th etc.) is lost. Table 19 illustrates how the current implementation handles the scenario and Table 20 shows the case of the proposed implementation.

Trig. Seg.	ACKS	First Block		Second Block		Third Block		Forth Block	
		L edge	R	L	R	L	R	L	R
5000	5500								
5500	(Lost)								
6000	5500	6000	6500						
6500	(Lost)								
7000	5500	7000	7500	6000	6500				
7500	(Lost)								
8000	5500	8000	8500	7000	7500	6000	6500		
8500	(Lost)								
9000	5500	9000	9500	8000	8500	7000	7500	6000	6500
9500	(Lost)								
10000	5500	10000	10500	9000	9500	8000	8500	7000	7500
15000	(Lost)								

Table 19. Case with the current implementation

Trig Seg.	ACKs	First Block		Second Block		Third Block		Forth Block		Fifth Block	
		R	L	R	L	R	L	R	L	R	L
5000	5500										
5500	(Lost)										
6000	5500	6500	500								
6500	(Lost)										
7000	5500	7500	500	1000	1500						
7500	(Lost)										
8000	5500	8500	500	1000	1500	2000	2500				
8500	(Lost)										
9000	5500	9500	500	1000	1500	2000	2500	3000	3500		
9500	(Lost)										
10000	5500	10500	500	1000	1500	2000	2500	3000	3500	4000	4500
15000	(Lost)										

Table 20. Case with the proposed implementation

It can be seen that, in the current implementation, by the 9th segment (sequence number 9000), 34 byes (2 + 4*8) would be used up by sending information of four blocks. When the 11th segment is received, information about the latest four blocks could be sent, losing information about the 6000 - 6500 segment. At the same time for the proposed implementation (Table 20):

$X = \log_2 4500 = 13$

Bits used up = 8 (kind) + 8 (length) + 8 (X) + 32 (sequence number for A) + 9*13 (offsets) = 173 bits = 22 bytes. Just 22 bytes of the available 40 have been used.

The improvement is more pronounced when TCP has to use the timestamp option along with SACK. Then SACK can convey information about only three blocks, and problem is encountered at the 9th segment itself, for the current implementation. However, if the proposed implementation is used:

$X= \log_2 3500= 12$

Bits used = 16 (timestamp) + 8 (kind) + 8 (length) + 8 (X) + 32 (sequence number for A) + 7*12 (offsets) = 156 bits (20 bytes). 20 bytes are still felt.

5.4 Simulation Results

To verify the performance enhancement brought about by the suggested alternate SACK option structure, simulations of various worst-case scenarios were carried out.

5.4.1 Experiment 1

In order to demonstrate the potential of the proposed implementation in resolving the problem faced by the current SACK implementation, we simulated the environment described in Table 19 using NS2 and observed the transmission of

packets through the network. The “List” error model of NS2 for the packet corruption was used to simulate the lossy environment.

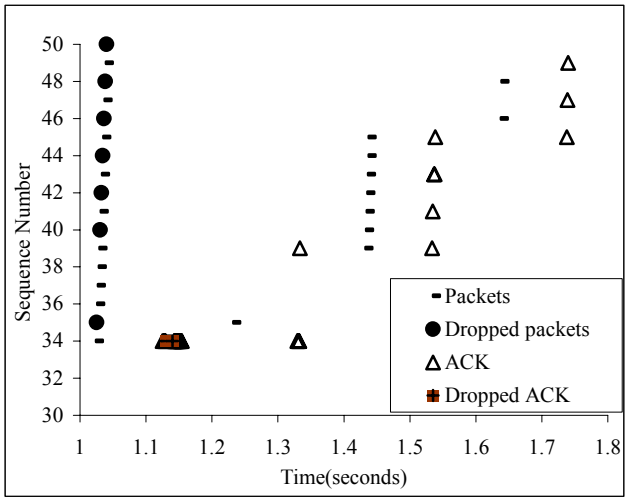

Fig. 39. Performance of original SACK

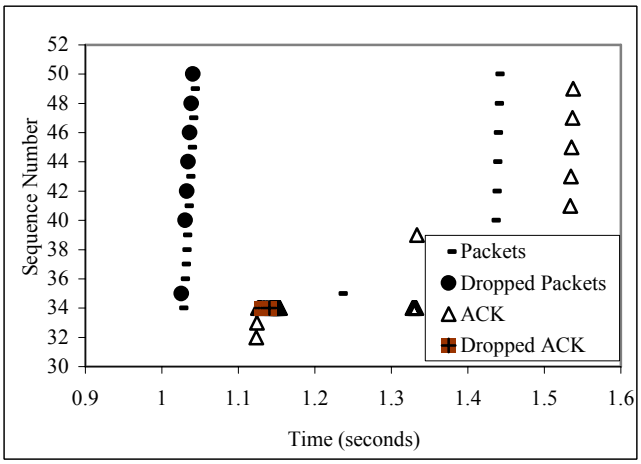

Fig. 40. Performance of modified SACK

It can be seen in Figure 39 that, because of the limitation of SACK, packets with sequence numbers 39,41,43 and 45 are unnecessarily retransmitted, as shown by two data packets being sent with the same sequence number at different time. However in Figure 40, these packets are not retransmitted. This shows clearly that the modified SACK does perform better than the present implementation.

5.4.2 Experiment 2

As presented in [31], unnecessary retransmission of packets can decrease the throughput of TCP connections. Thus the problem with TCP SACK that we see graphically in Figure 39 can lead to lower throughout. This is investigated in this experiment.

Performance gain obtained when file transfer was performed for a specified period of time was measured. The two-state Markov error model of NS shown in Figure 41 was used to generate the error conditions.

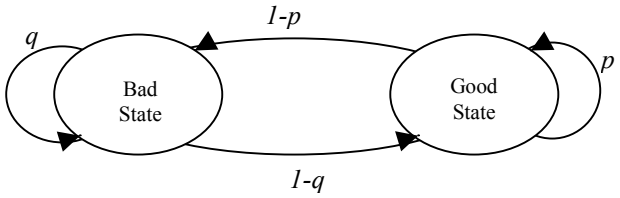

Fig. 41. Two-state Markov model for the lossy link

In this model, the link state is sampled every t_1 seconds when it is in the 'good' state and every t_2 seconds when it is in the 'bad' state. If the link is in a good state (respectively bad state) at the current sampling instant, then with probability p (respectively q) it will continue to be in the 'good' state (respectively 'bad' state) at the next instant also and with a probability 1-p (respectively 1-q) the transition to the 'bad' state (respectively 'good' state) takes place. When the link is in the 'good' state (respectively 'bad' state), each packet may be corrupted with the probability α (respectively β). The values used in the simulation are given in Table 21.

Variable	Source to Destination	Destination to Source
t1	25.0	25.0
t2	5.0	15.0
P	0.05	0.45
q	0.55	0.45
α	0.05	0.05
β	0.25	0.75

Table 21. Parameter values used in two state error model

Table 22 below shows the throughput measured for different periods of trial runs. Each value in the table is the average of measured throughputs from ten runs.

	50s	60s	120s	300s	600s
No-Offset (Kbps)	162.64	141.52	72.4	93.44	56.64
Offset (Kbps)	169.84	156.0	124.24	94.48	72.72

Table 22. Throughput over different trial run time periods

The above results clearly show that for long-lived TCP connections, under the error conditions simulated, the suggested modification to SACK performs better than the existing implementation. It can be seen that the new implementation of SACK always gives a better throughput.

5.5 Summary

Current SACK implementation has the limitation of being able to send a maximum of only 3 or 4 SACK blocks with each ACK. In this chapter of the thesis an alternate representation for the SACK blocks is proposed to overcome this limitation. Using examples and simulations, it was shown that the modified implementation of SACK produces better TCP performance in terms of the throughput obtained, and makes the SACK mechanism more resilient to the high packet error rates often seen in wireless scenarios.

Chapter 6

CONCLUSIONS

6.1 Summary

TCP Vegas was proposed to go beyond the earlier work on TCP congestion control. Its performance was seen to be better than TCP Reno in terms of throughput and retransmissions. However, various problems associated with TCP Vegas have been identified. These deficiencies have been a deterrent in using TCP Vegas widely in the Internet. In this thesis we examined the problems of TCP Vegas in detail and proposed a new algorithm (called TCP Vegas-A). The main idea of the new algorithm is that instead of assigning static values for the protocol parameters α and β, they are allowed to change in real time, allowing the connection to utilize the available bandwidth fully.

It was demonstrated using simulations in wired and satellite environments (GEO and LEO) that the proposed modifications do solve most of the problems associated with TCP Vegas. It was shown that in a wired environment, Vegas-A performs better than Vegas when competing with other TCP connections such as New Reno for shared bandwidth. It also overcomes the re-routing limitations of Vegas and is able to adapt faster to the changes in RTT and routes. It was observed that Vegas-A connections do not suffer from the unfairness towards old connection and unfairness against higher bandwidth connections.

In a satellite environment, it was shown that TCP Vegas-A performs better than Vegas. It was shown that Vegas-A is able to compete against New Reno more successfully than Vegas, while still retaining all advantages of Vegas. Simulations

proved that Vegas-A could handle the varying RTT behaviour of LEO satellite systems, while Vegas react badly to such fluctuations.

Thus, it is felt that adopting the Vegas-A modification would increase the chance of TCP Vegas-A being more widely used in the Internet and be a feasible alternative to the present standard TCP New Reno.

This thesis also looked into TCP SACK option and its worst-case limitation. Current SACK implementation has the limitation of being able to send a maximum of only 3 or 4 SACK blocks with each ACK. In this thesis, an alternate representation for the SACK blocks was proposed to overcome this limitation. Using examples and simulations, it was shown that the modified implementation of SACK produces better TCP performance in terms of the throughput, and makes the SACK mechanism more resilient to the high packet error rates often seen in wireless scenarios.

As a part of the experiments related to the thesis, it was shown that TCP SACK does perform better than New Reno in long latency error and congestion prone environment. However, there is a limit to which SACK can be helpful. It was shown that SACK has no effect when the level of corruption or congestion is very high.

6.2 Review of Thesis Objectives

In the first chapter, objectives of the thesis and the related research work were laid down as:

- Study the performance of various TCP congestion avoidance, control and retransmission mechanisms over wired and satellite networks.

- Study the limitations of TCP Vegas in detail.
- Investigate the dynamics of TCP Vegas and suggest changes to the protocol to resolve the limitations observed.
- Study the working of TCP SACK and its limitations.
- Suggest changes to the TCP SACK's current mechanism to resolve the limitations.

Extensive study was done on the performance of TCP New Reno and TCP SACK congestion control and retransmission algorithm on satellite (actual and emulated) and wired links. It was shown that TCP SACK performs better than New Reno in long latency error and congestion prone environment, but that SACK has no effect when the level of corruption or congestion is very high.

The limitation associated with TCP Vegas protocol was studied and verified using simulations. To overcome these problems, TCP Vegas-A, a modification to TCP Vegas, was proposed and through simulations, it was shown that Vegas-A did indeed solve most of the problem identified with TCP Vegas.

Then, limitation of TCP SACK under extreme error conditions was studied. A change to the implementation of the SACK options was proposed. Through simulations it was proved that the proposed change was able to overcome the identified problem and that the connection using the modified SACK was more resilient to errors in the network.

6.3 Future Work

Future work would involve testing TCP Vegas-A in the real network. This would involve the implementation of TCP Vegas-A modification in TCP Vegas code. In certain simulated scenarios it was seen that Vegas-A exhibited signs of instability. It is the belief of the author that this can be addressed by introducing a scaling factor in the feedback loop of the system. Instead of increasing the values of α and β by 1, one can use a weighted value of RTT or throughput change. It might also be interesting to look into the effect of changing the values of α and β independent of each other.

Work can also be done to determine the performance of TCP Vegas and Vegas-A over wireless networks like 802.11. As TCP Vegas relies on RTT to estimate the congestion in the network, fluctuating RTT, a characteristic of some wireless networks, may have a detrimental effect on the performance of TCP Vegas. However, it is the belief of the author that Vegas-A would be able to solve most, if not all the problems that might arise from such a scenario.

REFERENCES

[1] J. Postel, “Transmission Control Protocol”, RFC 793, IETF, September 1981.

[2] Sean McCearny and K. Claffy, “Trends in Wide Area IP Traffic Patterns, A View from Ames Exchange”, http://www.caida.org/outreach/papers/AIX005, May 2002.

[3] V. Jacobson, “Congestion Avoidance and Control”, Proceedings of SIGCOMM’88 Symposium, pp 314- 322, August 1988.

[4] V. Jacobson, “Congestion avoidance and control”, Computer Communications Review, 18(4): 314-29, August 1988.

[5] S. Floyd, T. Henderson, “The NewReno Modification to TCP’s Fast Recovery Algorithm”, RFC 2582, IETF, April 1999.

[6] L.S. Brakmo, S. O’Malley, and L.L. Peterson, “TCP Vegas: New techniques for congestion detection and avoidance”, Computer Communications Review, Vol. 24, No. 4, pp 24-35, October 1994.

[7] J.S. Ahn, P. Danzig, Z. Liu, and L. Yan, “Evaluation of TCP Vegas: Emulation and Experiment”, Proceedings of ACM SIGCOMM’95, pp 185-195, August 1995.

[8] Jeonhoon Mo, Richard J. La, Venkat Anatharam, Jean Walrand, “Analysis and comparison of TCP Reno and Vegas”, Proceedings of IEEE INFOCOMM’99, pp 1556-1563, March 1999.

[9] Richard J. La, Jean Walrand, and Venkat Ananthram, “Issues in TCP Vegas”, http://www.path.berkeley.edu/~hyongla, July 1998.

[10] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgment and Options”, RFC 2018, IETF, October 1996.

[11] Hans Kruse, “Performance of Common Data Communication Protocols over Long Delay Links: Experimental Examination”, 3rd International Conference on Telecommunications Systems Modelling and Design”, 1995.

[12] J. Postel, “File Tranfer Protocol (FTP)”, RFC 959, IETF, October 1985.

[13] J. Postel, “Telnet Protocol Specification”, RFC 854, IETF, May 1983.

[14] J. Postel, “Internet Protocol”, RFC 791, IETF, September 1981.

[15] M. Allman, S. Floyd, and C.Patridge, “Increasing TCP’s Initial Window”, RFC 2414, IETF, September 1998.

[16] Fall, K and Floyd. S, “Comparison of Tahoe, Reno and SACK TCP”, ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z, December 1995.

[17] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, “An extension to the Selective Acknowledgment (SACK) Options for TCP”, RFC 2883, IETF, July 2000.

[18] M. Allman and H. Kruse, “A History of the Improvement of Internet Protocol Over Satellites using ACTS”, Proceedings of ACTS Conference, 2000.

[19] M. Allman, D. Glover, NASA Lewis, and L. Sanchez, “Enhancing TCP Over Satellite Channels Using Standard Mechanisms”, RFC 2488, IETF, January 1999.

[20] M. Allman, et al., “Ongoing TCP Research Related to Satellites”, RFC 2760, IETF, February 2000.

[21] Hasegawa, K. Kurata, and M. Murata, “Analysis and Improvement of Fairness between TCP Reno and Vegas for Deployment of TCP Vegas to the Internet,” Proceedings of the IEEE International Conference on Network Protocols (ICNP 2000), November 2000.

[22] U. Hengartner, J. Bolliger, and Th. Gross, "TCP Vegas Revisited," Proceedings of IEEE INFOCOM '2000, pp. 1546-1555, March 2000.

[23] Steven Low, Larry Peterson, and Limin Wang, “Understanding TCP Vegas: A Duality Model,” Proceedings of ACM SIGMETRICS 2001, pp. 226-235, June 2001.

[24] S. Floyd, “Issues with TCP SACK”, Technical Report, LBL Network Group, 1996.

[25] V. Jacobson, R. Braden, and D. Borman, “TCP Extensions for High Performance”, RFC 1323, IETF, May 1992.

[26] Distributed Application Support Team, “Iperf”, http://dast.nlanr.net/Projects/Iperf

[27] Joseph Shaw, “Tcpdump”, http://www.tcpdump.org

[28] S. Ostermann, “Tcptrace” http://www.tcptrace.org

[29] Alessandro Rubini, “Rshaper”, http://ar.linux.it/software/index.html#rshaper

[30] Network Simulator (NS), available at http://www.isi.edu/nsnam/ns

[31] Reza Rejaie, Mark Handley, and Deborah Estrin, “RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet,” Proceedings of IEEE INFOCOM’99, pp.1337- 1345, March 1999.

[32] A. Romanow and S. Floyd, “Dynamics of TCP traffic over ATM Networks”, IEEE Journal on Selected Areas in Communications, Vol. 13 No. 4, p 633-641, 1995.

Papers published related to the thesis

- K.N. Srijith, Lillykutty Jacob, A.L. Ananda, "TCP Vegas-A: Solving the Fairness and Rerouting Issues of TCP Vegas", Proc. of 22nd IEEE International Performance, Computing, and Communications Conference (IPCCC) 2003, pp. 309-316, Phoenix, Arizona, USA, 9 - 11 April 2003.
- K.N. Srijith, Lillykutty Jacob, A.L. Ananda, "Worst-case Performance Limitation of TCP SACK and a Feasible Solution", Proc. of 8th IEEE International Conference on Communications Systems (ICCS) 2002, pp. 1157-1161, Singapore, 25 - 28 November 2002.
- Lillykutty Jacob, K.N. Srijith, Huang Duo, A.L. Ananda, "Effectiveness of TCP SACK, TCP HACK and TCP Trunk over Satellite Links", Proc. of IEEE International Conference on Communications (ICC) 2002, pp. 3038-3043, New York, USA, 28 April – 2 May 2002.
- K.N. Srijith, Lillykutty Jacob, A.L. Ananda, “TCP Vegas-A: Solving the Issues of TCP Vegas”, proposed for submission to Computer Networks, Elsevier Publications.