A Study on Transport Layer Congestion Controls

A thesis submitted in the partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Manoj Dahal

(*Registration No. 030 of Year 1995*)

School of Engineering
Department of Computer Science and Engineering
Tezpur University
Napaam, Assam, India

February, 2007

CERTIFICATE

This is to certify that the thesis entitled ***A Study on Transport Layer Congestion Controls*** submitted to the Tezpur University in the Department of *Computer Science and Engineering* under the School of *Engineering* in partial fulfillment for the award of the Degree of Doctor of Philosophy in *Computer Science* is a record of research work carried out by *Mr. Manoj Dahal* under my personal supervision and guidance.

All helps received by him from various sources have been duly acknowledged.

No part of the thesis has been reproduced elsewhere for award of any other degree.

Date: 22-7-2018

Professor D. K. Saikia

Place: Tezpur

Department of Computer Science & Engineering

School of Engineering

TEZPUR UNVERSITY

This is to certify that the thesis entitled ***A Study on Transport Layer Congestion Controls*** submitted by *Mr. Manoj Dahal* to the Tezpur University in the Department of *Computer Science and Engineering* under the School of *Engineering* in partial fulfillment of the requirements for the award of the Degree of Doctor of Philosophy in *Computer Science* has been examined by us on _____ and found to be satisfactory.

The committee recommends for the award of the degree of Doctor of Philosophy.

Principal Supervisor

External Examiner

Date:_____

ACKNOWLEDGEMENTS

With immense pleasure I would like to acknowledge the tremendous support and guidance I have received from the people who have helped me out during my research work.

This research-work is a team effort in which my supervisor has significant contribution and without whom I could not have dreamt of seeing this thesis a reality. My sincere thanks and heart-felt gratitude goes to my supervisor Professor Dilip Kr. Saikia who bailed me out during every problem I faced and who will be there in my memories forever for the immense help and guidance I received from him.

I would also like to express my sincere thanks to the faculty members and the staff of the Department of Computer Science and Engineering, Tezpur University who have provided help and support during my visits to the department. My special thanks go to Prof. Malaya Dutta, Prof. D. K. Bhattacharya, Dr. S. K. Sinha and Dr. R. K. Das. I would also like to express thanks to Mr. Dhiraj Sarma, Mr. Ajay Sharma and other members in the department who have helped me in accessing the computer laboratory and other facilities.

I would also like to express my gratitude to my previous employer National Informatics Centre (NIC) and my current employer Satyam Computer Services Ltd. who allowed me to stay in leave when I needed for the research work. My special thanks go to Mr. Birendra Chettri of NIC.

I would also like to express heart-felt gratitude to my loving wife Aruna and daughter Diva who endured the time I was away from them for my research work. And finally I sincerely acknowledge the love, encouragements and blessings flowered on me by my parents, in-laws, relatives, friends and our beloved Guruji that helped me to keep going during the tough times.

Manoj Dahal

Abstract

In this thesis we have studied the congestion control mechanisms in two reliable transport protocols, namely, TCP and SCTP. We have analyzed causes of the well reported lacunae in the congestion control mechanism of the TCP protocol, namely-

1. TCP's driving the network into congestion causing long queuing delays, packet loss and retransmissions that lead to high packet latencies, large delay jitters, and overloading of the network with retransmission traffic.
2. Considering any packet loss to be due to congestion and thus making the protocol unsuitable for networks with wireless links, where packet losses due to causes other than congestion are very common.

Based on the analysis we realize that TCP's driving the network into deep congestion, as a part of the probing for estimating the available bandwidth for the connection that leads to both these problems. We have therefore devised a new congestion control mechanism called *RTT based congestion control (RBCC)* for TCP that avoids driving the network into deep congestion so as to overcome these problems.

We have developed adaptations of this new algorithm for both wired networks as well as networks with wireless links. These adaptations have been implemented in NS2 environment and tested for comparison of their performance with respect to existing prominent variants of TCP, namely, Newreno, Vegas, SACK, and Snoop. Both these adaptations perform significantly better than the existing schemes in terms of lower packet loss, lower packet latency, lower delay jitter, and higher fairness and that without any compromise in the throughput. In the new scheme it is also possible to have control over packet latency and delay jitter with a marginal tradeoff on throughput.

As SCTP copies the congestion control mechanism of TCP it inherits the problems in TCP due to these mechanisms. We have therefore developed an adaptation of the new congestion control scheme for SCTP too. This adaptation referred to as *RTT based congestion Avoidance (RBCA)* has been implemented in NS2 environment and tested its performance in comparison to the standard SCTP that incorporates SACK and Delayed

ACK as built-in mechanisms. The simulation based experiments show similar improvements in the performance in SCTP with RBCA, in place of the existing mechanisms, as in the case of TCP.

It has been observed that SCTP does not make full utilization of its multihoming feature. It maintains an alternate path but uses it only for the retransmission of lost packets. We therefore devise another congestion control mechanism for SCTP, referred to as *Switch Path on Congestion (SPC)* that makes a balanced use of both, the primary as well as the alternate path. This mechanism can be used in SCTP along with RBCA as well as with standard SCTP. Simulation based experiments in NS2 environment show significant improvement in the performance of SCTP in both these cases in terms of packet loss, throughput, packet latency, and delay jitter. The new schemes also make SCTP much more robust to the high bit error rates prevalent in wireless environment.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

With the exponential nature of growth in the popularity of the Internet, the computer networks have become part and parcel of our day-to-day lives. It has become impossible to think of a world without the networks as information is a prime need of our lifestyle. These networks carry the information between host computers all over the world and at times beyond it. We have become so much dependent on these networks, particularly the Internet, that any disruption in the Internet today causes a shock wave in the business and commercial world as well as in the other spheres of human activity. Therefore proper, undisruptive functioning of the Internet is imperative to the well being of the world community, be it in the field of commerce, industry, education, governance or for that matter even entertainment.

The Internet, which is the inter-network of diverse networks spread all over the world, has TCP/IP protocol suite as base technology [Com03, Ste01, For06]. Like the OSI model the TCP/IP protocol suite is also layered. Some of the most prominent protocols in this suite are IP, TCP, UDP, and SCTP. The Internet Protocol (IP) at the network layer binds the networks together to create the Internet. IP is built on the principle of packet switching. It provides un-guaranteed, connectionless best effort service for packet delivery as the underlying networks are of diverse characteristics. The Transmission Control Protocol (TCP) at the Transport Layer is the most widely used transport protocol for the data transport service on the Internet. It makes the Internet connection oriented and reliable by adding flow control and error control mechanisms.

In a packet-switched network packets are stored in the intermediate nodes in the network before being forwarded to the next node on the path towards the destination. Packets from different source nodes may arrive at a node destined for different hosts. Depending upon the destination host these packets arriving at a node have to be forwarded along different outgoing links from the node. Each link has a capacity of carrying packets at a given data rate. If the arrival rate of packets at a node to be forwarded along a given link is less than the capacity of the link the packets get forwarded without delay. However if the arrival rate of packets at the node exceeds the carrying capacity of the outgoing links, the packets accumulate in the node. The packets then get queued up in the node in its memory buffers. The size of the buffer decides the maximum possible length of the queue. This phenomenon of accumulation of packets in the intermediate nodes is referred to as congestion [GCKB01, Tan99, WC91].

When packets accumulate at a node and the buffers get filled-up it starts affecting the nodes along the path backwards towards the source nodes with link-level back pressure pause mechanism [LPPBAB06, STJ02]. Packets start to accumulate in these nodes too as the nodes ahead stop accepting packets. Similarly, packets may start to accumulate in the nodes ahead too as the congested nodes send more packets to these nodes. This leads to spreading of the congestion to the rest of the network, or a part of it, with the originating node as the centre of the congestion [LPPBAB06]. The congestion in a network may be local or may become global depending on the extent of the spreading of congestion and the number of centres of congestion occurring at a time.

The result of congestion is that the utilization efficiency of the links leading to the congested nodes fall. That leads to drop in the carrying capacity of the network. During a congestion situation the queues in the nodes become long. The packets have to spend longer time in the queues before being forwarded to the next node along the path. The queuing delay suffered by the packets and thus the latency of the packets increase. Another important consequence of congestion is that when the buffers in a congested node become full, and the node becomes unable to accept any more packets. It then

starts dropping the subsequent packets forwarded to it. Packets are also dropped to create a path for the packets so as to recover from a congestion situation. The dropping of the packets leads to retransmission which causes additional traffic burden on the network and contributes further to the packet latency.

Congestion in a packet switched network thus leads to the following:

i. Drop in throughput of the network
ii. High packet latency
iii. Increased variation in the packet latency, i.e. jitter
iv. Packet loss
v. Additional load on the network due to retransmissions.

Thus occurrence of congestion affects the performance of applications to the extent of making the network unsuitable for many applications. Therefore it is very important to have proper control over the congestion situation in a network so as to maintain its suitability towards applications as well as for efficient utilization of the network capacity. The mechanisms that are deployed for detection, avoidance and recovery of congestion are called *congestion control* [RRQ03, Tan99].

The congestion control mechanisms can be built into one or more of the layers in the network [WC91]. The network layer can implement congestion avoidance and congestion detection mechanism and can help the upper layers in congestion recovery. *Traffic Shaping* [Liu92, VSK06], *Adaptive Routing* [FRT02, XQYZ04, AA03], *Resource Reservation* [SP98], *Weighted Round Robin Scheduling (WRR)* [Ray99, FH98], *Active Queue Management (AQM)* [BRHG04, RRQ03] etc. are some of the techniques that can be implemented at the routers for congestion avoidance. Schemes such as *Random Early Detection (RED)* [FJ93], *Explicit Congestion Notification* [RF99, YSL03] and *eXplicit Congestion control Protocol (XCP)* [KHR02] have been implemented with the help of the network layer for congestion detection and recovery.

One common technique used at the network layer to assist in congestion recovery is *packet dropping* [GCKB01]. Once the congestion is detected by dropping some packets at the routers resources can be freed to some of the flows so that the traffic movement can resume and process of recovery from the congestion situation can start. However without assistance from an end-to-end host layer in the way of reducing the rate of packet injection into the network, the network layer alone cannot achieve full control over congestion. The transport layer is the lowest end-to-end layer having full control on the flows over the connections. Therefore it is suitable for implementing full fledged congestion control mechanisms. For this, however, it has to maintain the state of the connection and has to have a feedback mechanism. On the Internet the congestion control mechanisms are built into the transport layer protocols.

1.2 Internet Data Transport

As has been said, the Internet embraces diverse networks with wide range of technologies and communication characteristics and these are bound together by IP [Com03, Ste01, For06]. In doing so it is unable to provide guaranteed packet delivery service. However, the end user application needs some robust transport mechanism for transferring user data without worrying about the underlying technologies. Internet Data Transport (IDT) provides mechanisms by which Internet applications can be provisioned with different kinds of services the user needs. These services are divided broadly into two categories. One is reliable delivery of user data and second is the unreliable but timely delivery. Most of the Internet applications need either of these two kinds of services. Reliability is provided by maintaining connection state across source and destination computers, retransmitting lost data, having congestion control, flow-control and buffering. The reliable transport service ensures that a user message reaches its destination in-order giving less attention to the amount of time it takes. It uses flow-control techniques to prevent overflow in the buffer space at the receiver and congestion-control mechanisms to control the rate of injection of packets into the subnet.

The second type of transport service, on the other-hand, does not ensure reliable and ordered delivery of messages. But it provides timely delivery of messages no matter if some of packets are lost in the middle. Real-time applications that need data-packets to be delivered with consistent inter-arrival time use this type of service.

The principal protocols used for data transport over the Internet are- the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP). The Stream Control Transmission Protocol (SCTP) is a newer transport protocol that has attractive feature and has attracted attention of researchers as well as the industry [ISAS03, CIALHS03, FA04]. TCP is currently the most widely used transport protocol. It provides reliable byte-stream service by ensuring that each data-byte sent from source reaches the destination without fail [Com03, Ste01, For06]. It does so by establishing connection between the application processes in the two hosts, having provision for acknowledgement for each data-packet from receiver to the sender and having a retransmission mechanism for lost or erroneous data-packets. TCP maintains two window variables viz. receiver window (*rwnd*) and congestion window (*cwnd*). The first one (i.e. *rwnd*) provides flow control which makes the sender not to exhaust receiver's buffer space. The *cwnd* variable allows it to control packet injection rate and thereby have control over congestion situation in the subnet.

UDP on the other hand provides unreliable transport service [Com03, Ste01, For06]. It is a datagram oriented protocol without much overhead. Each output transport operation by an application process produces exactly one UDP datagram, which causes one IP packet to be sent. UDP does not guarantee the delivery of a datagram. The datagram's arrival at the destination depends upon the conditions in the subnet. Also, UDP does not have any congestion control mechanism. It injects the packets into the subnet at whatever rate an application produces them. Therefore an application process using UDP may need to have its own congestion control mechanism.

Stream control transmission protocol (SCTP) is the new evolving technology for Internet Data Transport [Ste00]. It has embraced all the features of TCP and added some new features to support next generation Internet applications. The new features supported in SCTP primarily are- multi-homing, multi-streaming, message boundaries and mechanism for protection against denial-of-service-attacks. The congestion control mechanisms in SCTP are similar to those in TCP. With the multi-homing feature it allows multiple paths between a source and destination pair. SCTP controls congestion for each of such parallel paths independently.

TCP, UDP and SCTP use port numbers in their headers to multiplex more than one application. This enables a host to maintain several network connections simultaneously in spite of having a single IP address.

As UDP does not have a mechanism for congestion control, the study focuses on the other two transport protocols, namely, TCP and SCTP. There are several issues related to congestion control that affect the overall performance of the network and need close scrutiny.

1.3 Measure of Performance for Congestion Control Scheme

The performance of a congestion control scheme is measured in terms of several parameters. These parameters are- throughput, packet loss, fairness, packet latency, jitter and queue occupancy in the bottle-neck link.

Throughput: Throughput is the total volume of data successfully delivered at the receiving hosts over a given time. It can be expressed in terms of the total number of bytes delivered. It can also be expressed in terms of capacity utilization as percentage of the total carrying capacity of the subnet.

Packet loss: Packet loss is the total number of packets that is not received by the destination node. The loss of packets could be due to either traffic overload at the bottle-neck link that leads to packet dropping or due to link error (e.g. signal fading). While measuring the performance of a congestion control scheme, only packets lost due to traffic overload (i.e. network congestion) is taken into account.

Fairness: In a network each flow should get a fair share of the bandwidth. It should not be such that some flows hog most of the bandwidth while the others starve. This can happen if some of the flows while adjusting the individual congestion windows set them higher than appropriate. By fairness, one can measure how fairly the flows that are competing for network resources get their share. If the share of the bandwidth enjoyed by the flows are unequal then the mechanism is poor in fairness. If there are ten flows that are competing for same network resources with similar characteristics of network configurations then each of them should ideally get 10% of aggregate resource. The fairness index can be computed as follows [CJ89]:

$$F = (\Sigma x_i)^2/n(\Sigma x_i^2) \quad \text{---------- (1.1)}$$

Where x_i is the throughput for i-th flow.

The fairness index above is bound between 0 and 1, which is a continuous function. Greater the value of F (i.e. near to 1) the better is the scheme in fairness. Fairness can also be expressed in percentage as-

$$F = (\Sigma x_i)^2/n(\Sigma x_i^2)\ X\ 100\% \quad \text{---------- (1.2)}$$

Packet Latency: Packet-latency is the total amount of time taken by a packet for traversing from its source to the destination host. Packet latency accounts for the transmission time, propagation time, queuing delay and the processing time at the routing nodes.

Jitter: Delay jitter or simply jitter is the variation in the packet-latency that occurs on a connection. It is normally expressed as the standard deviation of the packet latencies experienced on the connection. Jitter has affects on the performance of real-time applications requiring connection oriented service. The effect of jitter can be reduced by using memory buffers at the receiver. However the buffer size requirement grows with jitter.

1.4 Motivation Behind the Work

TCP [Pos81, Ste01, Com03] is the most widely used transport protocol in the Internet. Its congestion control mechanism has a significant role to play in the overall performance of the network. TCP does not take help of the lower layers to determine the availability of bandwidth for a new connection. Therefore it follows a policy of probing. This probing consists of increasing the packet injection rate for a flow until there is a congestion. When a congestion occurs it reduces the injection rate. It employs additive increase and multiplicative decrease in the rate of injection of packets into the network by increasing and decreasing the congestion window at the source additively and multiplicatively respectively [Pos81, Ste01, Com03]. The increase is done when the acknowledgements (ACKs) for the packets sent are received regularly. When congestion occurs at a router, packets get dropped resulting is non-receipt of ACKs at the sender. The sender then times out and interprets this non-receipt of ACKs within the time-out duration as loss of packets due to congestion in the network. It then reduces the packet injection rate by reducing the congestion window size by a factor. The source also then

retransmits packets that are presumed to be lost. In this scheme congestions are bound to occur as the packet injection rate at the sending end is continued to be increased until congestion occurs to the extent that the routers start dropping packets.

This reduction in the congestion window size in TCP is not restricted to packet losses due to congestion as TCP cannot differentiate between packet lost due to transmission errors (such as interference, fading etc.) and congestion. In a wireless environment packet losses due to transmission errors are quite frequent. Reduction in congestion window size in such situation leads to loss in throughput for the connection.

TCP's dependence on packet dropping to detect congestion forces it to drive the network into congestion. As a result all the ill-effects of congestion are inherent in the TCP's congestion control mechanism. Since the congestion control mechanism in SCTP [Ste00] is borrowed from TCP, these ill-effects are inherited by SCTP too.

One way to overcome this problem is to avoid pushing the network into extreme congestion situation that causes packet drops. When congestion situation approaches the queue lengths at the routers start increasing. This increase in queue length causes the packets to suffer more delays at the routers. Finally when the queues become full the routers start dropping packets requiring retransmission of these packets. The retransmitted packets suffer excessive delays leading to the highly undesirable delay jitters. Retransmissions also put additional traffic burden on the subnet. As a result the throughput of the network also suffers. These inefficiencies have further detrimental effects on the QoS parameters such as high latency, high delay jitter and reduction in the packet throughput. To avoid these inefficiencies it is therefore necessary to control a congestion situation while it approaches rather than allowing congestion to reach the extreme stage of losing packets before starting the control mechanism.

1.5 The Work Done

In this thesis we present a new scheme for congestion control in TCP based on round-trip time (RTT) that maintains the RTT for a connection hovering around a threshold value. This new scheme, which is referred as *RTT Based Congestion Control (RBCC)*, avoids occurrence of extreme congestion situation and thus largely eliminates packet losses due to congestion. The scheme also provides control over packet latency.

We have devised adaptations of this scheme for wired and wireless networks. These adaptations of the scheme have been studied in simulated wired and wireless communication environments. The simulation studies [DS04, DS06I] show that RBCC has significant advantages over existing schemes such as TCP Newreno[Hoe95, FH99], TCP Vegas[BOP94] and SACK [MMFR96], in respect of reduction in retransmissions, packet latency, and delay jitter while maintaining the same or even higher levels of throughput in both of the wired and wireless environments. A comparative study with TCP Snoop[BSAK95] in Wireless Environment shows that the new technique performs better in terms of latency and fairness and in some situations better in terms of throughput.

Further, the work has been extended to congestion control in SCTP. Apart from this, a new technique called *Switch Path on Congestion (SPC)* [DS06II] has also been developed that uses the multi-homing feature of SCTP. Simulation study shows that that RBCC with as well as without SPC has significant advantages over the existing SCTP in terms throughput, packet-loss, fairness and latency.

1.6 Organization of the Thesis

The rest of the thesis is organized as follows. A survey of the existing works in the area and a discussion on the existing TCP congestion control mechanisms are presented in Chapter 2. In this chapter the SCTP protocol, its congestion control mechanism and the relevant issues are also discussed. Chapter 3 makes an analysis of RTT as a measure for congestion in the network and presents proposed RTT based congestion control scheme for TCP. Chapter 4 presents the scheme for wired network environment and the algorithms for setting the threshold RTT statically as well as on-the-fly. The results of simulated experiments in wired network environment showing the comparative performance of the new scheme in relation to TCP Newreno, TCP Vegas and TCP SACK are also presented in this chapter. Chapter 5 presents the adaptation of the scheme to the wireless environment and the experimental results that compare its performance with TCP Newreno, TCP Vegas, TCP SACK and TCP Snoop in a network with wireless links. Chapter 6 discusses how the scheme is adapted to the new transport protocol SCTP. It also presents the new scheme *Switch Path on Congestion (SPC)* that exploits the multi-homing feature of SCTP. Results of experimental comparative study of our schemes for SCTP with the existing ones are also presented here. Chapter 7 summarizes the work.

Chapter 2

A Survey of Transport Layer Congestion Control

TCP is the reliable transport protocols primarily used in the internet today. SCTP is a new and upcoming protocol with potential for being widely used on the internet in the near future, particularly for real-time applications. In this chapter a survey is made on the congestion control techniques in these two transport protocols.

2.1 Congestion Control in Transmission Control Protocol (TCP)

In the Internet, congestion control is the responsibility of the transport layer. TCP is the most widely used protocol for Internet Data Transport. Congestion control is therefore an integral part of TCP. For congestion control TCP adopts a congestion detection and recovery approach.

The prime function of TCP [Com03, Ste01, Pos81] is to ensure delivery of user data to the destination host application process. This is achieved through an acknowledgement mechanism in which the receiver sends an acknowledgement (*ACK*) to the sender on receipt of an error free packet. A timer is associated at the sender's end with each of the packets transmitted to keep track of packets for which *ACK*s are pending. If an *ACK* is not received for a packet within a time-out period the packet is assumed to be lost. The lost packets are then retransmitted by the sender. The time-out period is fixed based on the *Round-Trip Time (RTT). RTT* is defined as the time between the sending of a packet and the receiving of its *ACK* at the sender's end. The time out period is set to be a factor of times higher than the average *RTT* to avoid premature retransmissions.

TCPs general strategy is to keep the transport layer as independent of the other layers as possible. Therefore the TCP does not take any direct help from the lower

layers for detection of congestion. The congestion detection mechanism in TCP is based on detection of packet drops and any packet lost is interpreted as a packet dropped. Therefore whenever a packet is lost TCP assumes it to be due to occurrence of congestion along the path.

TCP does not get any information about the bandwidth availability for a connection from the other layers. But it is important that TCP utilizes the available bandwidth of the network to maximize throughput. To make an estimate of it TCP follows a probing strategy. It tries to obtain as much bandwidth for a connection as possible while being fair to the other connections. To achieve these objectives, TCP maintains a transmission window called the *Congestion Window (cwnd)* at the sender. The *congestion window* decides the number of bytes that can be injected into the network at a time and thus keeps a control over the bandwidth that is consumed by the connection. When a new connection is established, the starting *cwnd* is set to one segment, which is generally set to the maximum packet size also called as *Maximum Transmission Unit (MTU).* It then transmits a packet on receipt of which the receiver sends back an *ACK* packet. On receipt of this *ACK* it increases the *cwnd* size by one segment. Next time two packets (as the value of *cwnd* is then two segments) are sent and upon receiving two *ACK*s the *cwnd* is increased by two. Thus the successful receipt of *ACK* packets is interpreted by the sender as availability of bandwidth and doubles the congestion window size thereby doubling the consumable bandwidth for the connection. It continues doubling the *cwnd* size until it reaches a threshold level called the *Slow Start Threshold (ssthresh)* while the packets are successfully delivered at the destination. This phase of probing is known as the *Slow Start* phase. After the *cwnd* reaches this threshold level, instead of doubling it the *cwnd* size is increased by one segment per RTT on receipt of *ACK* for each successful delivery of packet at the destination. This phase of linear increment of the *congestion window* is known as the *Congestion Avoidance* phase. Though the rate is reduced in *Congestion Avoidance* phase, the increment of *cwnd* is continued until some packet loss is detected. This naturally leads to a rate of injection of packets by the sender into the network that is in excess of what the network can accommodate thus driving the network into congestion. This excess rate of injection results in dropping of packets at the bottle-neck links and non receipt of *ACK*s at the sender. The sender then times-out and interprets this *ACK* time-out as packet dropped due to occurrence of congestion. TCP then tries to overcome

the congestion situation by bringing down the congestion window size to its minimum (i.e. one segment) and starts the process of probing all over again with the *Slow Start* phase.

In the original version of TCP [Ste01] a packet-loss is detected when the timer goes off. Then the packet is retransmitted. The *Time-out* is set to be significantly higher than RTT. As a result at times this leads to long waiting time before the retransmission can be done. This affects throughput, causes high packet latencies and large delay jitters. To avoid these, the *Fast Retransmit* [Jac90, APS99] scheme uses the arrival of three or more duplicate *ACK*s in a row as an indication of loss of a packet. and the receiver then retransmits the missing packet without waiting for a retransmission timer to expire. After a packet-loss detection in this manner *Congestion Avoidance* is performed by setting *ssthresh* to half of the *cwnd* and then setting the *cwnd* value to *ssthresh*. By setting the *cwnd* value to *ssthresh* the *slow-start* phase is avoided. The scheme directly goes into the *congestion avoidance* phase. This modified scheme is a known as *Fast Recovery*. The use of the *Fast Recovery* scheme along with *Fast Retransmit* it is known as *TCP Reno* [Jac90, APS99].

TCP Reno however does not recover from multiple packet losses within a single window. When there are multiple packet losses, the acknowledgement for the retransmitted packet (the first one in Fast Retransmit) will acknowledge some but not all of the packets transmitted before the *Fast Retransmit*. In an improved scheme called *TCP NewReno* [Hoe95. FH99] the sender recovers from multiple packet losses by inferring receipt of partial acknowledgements as packet lost and retransmitting the indicated packets.

Multiple packet losses from a window of data can have a catastrophic effect on TCP throughput. With the limited information available from cumulative acknowledgments, a TCP sender can only learn about a single lost packet per round trip time (RTT). A Selective Acknowledgment (SACK) mechanism [MMFR96], combined with a selective retransmission policy, can help in overcoming these limitations. The receiver in this case sends back SACK packets to the sender informing it of data that has been received. The sender can then retransmit only the missing packets. The TCP FACK (Forward Acknowledgement) [MM96] improves congestion control during

recovery phase by providing accurate measure of outstanding data in the network. TCP FACK works in conjunction with TCP SACK.

2.1.1 Issues in Congestion Control in TCP

For all the above the basic mechanism remains the same in that the network is always driven into a deep congestion situation leading to packet loss before recovering from it. This results in several undesirable effects. As more and more packets are injected into the network the queue length in the switches start growing. This causes the packets to suffer longer delays and hence the packet latencies increase. Finally when the packets are dropped these packets are retransmitted. This puts extra burden on the network and lowers its effective capacity. The retransmitted packets also suffer large delays leading to excessive jitters.

Another serious drawback of the scheme is that it interprets any packet loss as packet drop due to congestion [BPSK97, BV98, CBDA98, GRL99, HK99A]. As a result a packet lost due corruption over a noisy link is also interpreted as a packet dropped due to congestion. This misinterpretation leads to unnecessary reduction in the congestion window and a corresponding fall in the throughput [TM02]. In a network, involving wireless links, packet loss due to noise is very common. In such environments the above scheme becomes very inefficient and unsuitable [BPSK97, BV98, CBDA98, GRL99, HK99A]. To counter this, Explicit Congestion Notification (ECN) [RF99] was proposed which takes the help of the network layer. In this scheme the routers are allowed to set the congestion experienced (CE) bit in the IP packet header as an indication of congestion to the end host rather than dropping the packet. Upon receiving *ACK* packets set with CE bit, TCP reduces the size of it's congestion window. One advantage of this is TCP can avoid the retransmission of packets that would have been dropped by the router. However, researcher have noted that ECN cannot be relied upon to completely eliminate packet drops as an indication of congestion and would not remove the danger of congestion collapse for best-effort traffic [Flo00].

2.1.2 Works on RTT/ Delay based Congestion Control in TCP

To overcome the above shortcomings it is necessary to avoid driving the network into extreme congestion situation and not use packet loss for detection of

congestion situation. One alternative is to use the Round Trip Time (*RTT*) or delay as a measure for the traffic load in the network and use it to estimate the available bandwidth. As *RTT* depends upon queuing delay at the routers and queuing delay is an indication of the level of traffic in the network it can be used as a measure for the traffic volume in the network. The following are some *RTT* or delay based congestion control schemes already proposed for TCP:

i) Jain's Work

In one of the earliest works based on delay, Jain [Jai89] proposed a scheme for congestion avoidance which observes the sign of the variation in *RTT* with respect to variation in the congestion window size. If an increase of the congestion window results in an increase in the RTT, then the congestion window size is decreased. Otherwise, the congestion window size is increased. The scheme defines a *normalized delay gradient* (NDG) to be used as a decision function as follows:

$$NDG = ((D-D_{old})/(D+D_{old}))((W+W_{old})/(W-W_{old}))$$

where D and D_{old} are the round-trip delays at the windows W and W_{old} respectively.

Based on the value of *NDG* change is applied to the window size once every two round-trip delay. If $NDG > 0$ or $W = W_{max}$ then the window is decreased multiplicatively by c. Otherwise, if $NDG \leq 0$ or $W = W_{min}$ then the window is increased linearly by ΔW. The recommended value for ΔW is 1 and for c it is 0.875. W_{min} is the minimum window size and W_{max} is maximum window size. The scheme is reported to able to keep the window around the knee point (the point where the throughput starts saturating) well.

ii) Tri-S Scheme

Tri-S [WC91] scheme uses a change in the throughput as an indication of congestion. The algorithm computes *normalized throughput gradient* (NTG) and compares to a threshold to keep the connection at the optimal operating point which maximizes the throughput while avoiding congestion. The scheme has three operation modes as described below:

a) When a user initiates a connection, it enters the initialization mode. The window size is set to one basic adjustment unit (BAU). Upon receiving each *ACK*, the *cwnd* is increased by one *BAU* until the *cwnd* is equal to the receiver window.

b) When a packet times out the *cwnd* is set to one *BAU* and increased by one *BAU* for each *ACK* while the *NTG* is above a certain threshold NTG_d.

c) In the increase mode, the *cwnd* is increased by BAU/(*current cwnd)* each time an *ACK* is received. If the accumulated increase is larger than the packet-size, the *NTG* is checked. If the *NTG* is less than the threshold NTG_d, the *cwnd* is decreased by one packet-size, otherwise do nothing.

It has been reported [WC91] that Tri-S performs better in comparison to the slow-start algorithm in terms of throughput and fairness. It also maintains lower queue length in the bottle-neck link than the slow-start.

iii) TCP Dual

TCP Dual [WC92] has the concept of delay threshold which is calculated as the average of the minimum and maximum RTT. For every two round-trip time, the round-trip delay is checked in *Slow Start* phase and if it is greater than the threshold then the *cwnd* size is reduced by 1/8 of the current size. TCP Dual resets the *cwnd* size, the minimum RTT and the maximum RTT values on timeout. It then updates the minimum and maximum RTT values on each round-trip time.

A study [WC92] shows that TCP Dual eliminates periodic packet losses and reduces oscillation in window adjustment and the queue length. However, for the initial period (until the first packet-drop) the performance of TCP Dual is similar to that of TCP Newreno.

iv) TCP Vegas

The best-known TCP congestion control scheme that uses RTT is TCP Vegas [BOP94]. Here, first an *Expected* throughput rate in the network is estimated as the size of the current congestion window divided by the minimum of all measured round-trip times since the establishment of the connection. Then, Vegas calculates the *Actual* throughput rate in the network by recording the number of bytes transmitted in a distinguished segment, determining the RTT for the segment

and dividing the number of bytes transmitted by the RTT. Two thresholds α and β, $\alpha < \beta$ are defined, roughly corresponding to having too little and too much extra data in the network, respectively. If the difference *Diff = (Expected – Actual)* is found to be less than α, the congestion window is increased linearly, and if *Diff* > β, the congestion window size is decreased linearly during the next RTT. The congestion window is left unchanged when α <*Diff* <β. Vegas also makes change in the retransmission mechanism. It uses timestamp for each packet sent to compute the round-trip time on each *ACK* received. When a duplicate *ACK* is received it checks to see if the difference between the timestamp and the current time is greater than the timeout value. If it is, Vegas retransmits the packet without waiting for the third duplicate *ACK*.

v) TCP Westwood

The basic idea of TCP Westwood [MCGLS00] is to exploit the stream of returning *ACK* packets in order to obtain an estimate of the available bandwidth for the connection. It estimates the available bandwidth in the path using number of packets acknowledged by an *ACK* packet and the time since last acknowledgement. Then it deduces the congestion window and *slow start threshold* based upon the estimated available bandwidth and the minimum RTT. The bandwidth estimate is used to properly set the *cwnd* and *ssthresh* when a congestion situation is detected. In absence of congestion situations, the dynamics of these variables conforms to that of standard TCP.

vi) TCP FAST

TCP FAST [JWL04] divides congestion control mechanism into four functionally independent components. These four components are *data control*, *window control*, *burstiness control* and *estimation* component. The *data control* component determines which packet to transmit, *window control* determines how many packets to transmit, and *burstiness control* determines when to transmit these packets. These decisions are made based upon the information provided by the *estimation* component. It uses both queuing delay and packet-loss as signal of congestion. TCP FAST updates the congestion window based on average RTT, the minimum RTT observed over the path since the beginning of the connection and the previous congestion window. It is reported [JWL04, JWL03] to demonstrate

encouraging performance enhancements.

vii) ***The work of Zhang and Tsaoussidis***

The work by Zhang et al. [ZT03] proposes a congestion avoidance mechanism by reducing *cwnd* size based upon queuing delay that makes use of RTT. This technique was proposed to improve the smoothness in the sending rate of real-time applications within the framework of bandwidth efficiency and fairness. The mechanism relies on fine-grained RTT estimation to measure the network condition and coordinates the upward and backward window adjustments to avoid the damage due to unsynchronized window control on throughput smoothness. The scheme complements TCP's standard congestion control algorithm specifically with two techniques. Firstly, if it finds *qdelay/max_qdelay* >= Th_{upper}, then it reduces the *cwnd* by a factor. Similarly, if it finds *qdelay/max_qdelay* <= Th_{lower} then it accelerates the *cwnd* increment by a factor. Initially it allows the bottle-neck queue length to grow to the fullest to observe maximum RTT with highest queuing delay. However it also reduces the *cwnd* on detection of packet-loss.

A comparison with Reno, Vegas, Reno/RED shows [ZT03] that the above technique displays better fairness and smoothness in throughput without sacrificing throughput.

viii) Other RTT/delay related works

An algorithm called Virtual Rate Control (VRC) proposed in [PLPC04] makes the use of RTT along with queue length for controlling the window-size. Similarly, in [MTZ05] authors claim that using an RTT-based mechanism the TCP sender can determine, with sufficient accuracy, the level at which the bottleneck-queue becomes full.

The above RTT and delay based techniques can be summarized in the following table 2.1:

Table 2.1: Summary of existing RTT/Delay based Schemes

Sl. No.	RTT/Delay based Scheme	Calculation of RTT	Use of RTT	Performance
1	*Jain's Work*	Round-trip delay on respective windows	If increase in *cwnd* results in an increase in the RTT then the *cwnd* is decreased otherwise *cwnd* is increased: this is done on every two RTT.	It is able to keep window below knee point
2	*Tri-S Scheme*	Normal RTT	The normal RTT is used to calculate throughput	Better than slow start algorithm in terms of throughput and fairness
3	*TCP Dual*	Calculates delay threshold as an average of minimum and maximum RTTs	On every two RTT it checks if the RTT is greater than the threshold then it reduces the *cwnd* by 1/8	Eliminates the periodic packet losses and reduces oscillation in queue length
4	*TCP Vegas*	Uses timestamp option in TCP packets to calculate fine-grain Base RTT (the minimum of all measured RTTs) and sample RTTs	The base and sample RTTs are used to compute expected and actual throughputs respectively.	Shows 40-70% better throughput than TCP Reno
5	*TCP Westwood*	Calculates minimum RTT based upon continuous monitoring of RTTs (on every ACK)	Uses minimum RTT to estimate bandwidth	Better throughput than SACK and Reno
6	*TCP FAST*	Calculates minimum RTT and average RTT	It uses minimum and average RTT to update the *cwnd*	Shows better capacity utilization than Linux TCP (also including RED)
7	*The work of Zhang and Tsaoussidis*	Does the fine-grain RTT estimation	It calculates queuing delay and maximum queuing delay using the RTT and then adjusts the *cwnd* depending upon the ratio of these two	Better fairness and smooth throughput than Reno, Vegas and Reno-RED

2.1.3 Works on wireless related issues in TCP:

In the literature, several solutions have been proposed for addressing TCP's problem over wireless networks. Some of these are discussed below:

i) Indirect-TCP

Indirect-TCP (I-TCP) [BB95] splits a TCP connection at the Base Station (BS) so as to completely shield the sender from the effect of wireless losses. The BS maintains two connections, one to the Fixed Host (FH) and another to the Mobile Host (MH). This technique helps in hiding the poor quality of wireless link from the FH. By splitting a connection, I-TCP doesn't maintain end-to-end semantics of TCP. When a packet arrives at the BS, it sends an *ACK* to the FH and the packet to the MH after caching. Afterwards, if the packet is lost between BS and MH links, BS itself retransmits. I-TCP acknowledgements are not end-to-end. Since the TCP connection is explicitly split into two distinct ones, acknowledgements of TCP packets can arrive at the sender even before the packet actually reaches the intended recipient. This results in a break in the semantics from the original TCP.

ii) Mobile TCP (MTCP)

The MTCP [BS97] is similar to I-TCP, but here last byte of data is acknowledged to the source only after it is received by the MH. Although the source falsely believes that every data-byte except the last byte is received by the receiver, it can take remedial action based on whether the last byte is received or not. In particular, if *ACK* for the last byte is not received by source then it has to resend all the data including those that may already have been received by MH. By holding the last byte, the BS can send zero window advertisement to freeze the source during handoffs, so that the window and timeout are not affected.

iii) Explicit Bad State Notification

In Explicit Bad State Notification (EBSN) [BKVP96] local retransmissions are made from the BS to shield the wireless link errors and to improve throughput. However if the wireless link is in error state for an extended duration, the source may timeout causing unnecessary retransmissions. The EBSN approach avoids

source timeout by using an explicit feedback mechanism. If the wireless link is in bad state, the BS sends an EBSN message to the source for every retransmission of a segment to the mobile host. The EBSN message causes the source to reinitialize the timer. If the BS sends the EBSN message before the source timer expires, then there will be no timeout at the source. However, the main disadvantage of this approach is that it requires TCP code modification at the source to be able to interpret EBSN messages as well as specific requirements at the BS.

iv) Wireless TCP

Wireless TCP (WTCP) [RM98] proposes to hide the time spent by a TCP segment in the Base Station (BS) buffer. Thus RTT estimate and timeouts maintained at the sender (and consequently sender's ability to detect wired congestion losses) are not affected by the wireless losses. This is achieved without an explicit feedback message, rather by modifying the timestamp field in the *ACK* packet. In WTCP, the TCP connection from source is terminated at the BS and, another reliable connection is established from BS to MH. However the BS acknowledges a TCP data segment to the FH only after getting *ACK* from MH. The reliable connection from BS to MH takes into account the unique characteristics of the local wireless link. In case of timeout, the transmission window for wireless connection is reduced to just one segment assuming a typical burst loss on the wireless link is going to follow. Unlike regular TCP, each time an *ACK* is received, WTCP opens the wireless transmission window completely assuming that an *ACK* indicates that wireless link is in good state. That is, when an *ACK* is received, the transmission window size is set to the receiver's (i.e. MH's) advertised window size.

v) TCP Snoop

Snoop [BSAK95] is similar to WTCP except it is implemented at link layer of the base station. The base station sniffs the TCP segments destined for wireless/ Mobile Host (MH) and buffers them if buffer space is available. The segments retransmitted by the Fixed Host that have already been acknowledged by BS are not forwarded to MH. The BS also sniffs the *ACK* packets from the MH. If it finds a duplicate *ACK* then it detects a segment loss and if the segment

is in the buffer then retransmits it to the MH and starts a timer. The duplicate *ACK*s are also dropped to avoid unnecessary fast retransmits at the fixed host. The Snoop also tries to hide wireless link related losses from the sending host. The requirement of sniffing the packets in the BS in snoop however leads to security problem. The sniffing is also not possible if encryption is used at the network layer as in IPSec.

vi) Space Communications Protocol Standards-Transport Protocol

Durst et al. in [DMT96] have done some work for similar kind of scenarios i.e. space communication. In that, they have formulated the Space Communications Protocol Standards-Transport Protocol (SCPS-TP). SCPS-TP is an extension of standard TCP with changes in some specifications that is designed particularly for the links where there is chance of frequent transmission errors and intermittent connectivity (i.e. link-outage). To deal with the transmission error, when a receiving ground station moves into link-corrupted state (i.e. when a threshold number of packets fail CRC), it begins sending *corruption-experienced* ICMP messages to the destinations contained in the cache. The destinations inform their respective SCPS-TP sources about the corrupted link via a TCP option on the acknowledgement segment. Similarly, to deal with link outage, a receiving ground station detects link outage by a loss of carrier lock or the received signal strength falling below some threshold. Then the ground station sends a *link-outage* ICMP message to any host on its own side of the severed link from which it receives traffic. It makes adjustments to TCP Vegas congestion control and window scaling mechanism, including detection of packet loss due to bit errors on error-prone links [GRL99]. It avoids making assumption that all packet loss is due to congestion rather than bit errors. SCPS-TP uses header compression and Selective Negative Acknowledgement (SNACK) to overcome link capacity limitations. SCPS-TP experimental results on NASA ACTS satellite link showed improved performances.

vii) MAC in IEEE 802.11

IEEE 802.11 [IEEE99] is the IEEE standard for Wireless LAN (WLAN) that provides the specifications for MAC and Physical Layers. One of the major MAC Layer functionalities provided in IEEE 802.11 is the automatic

transmission of acknowledgement frame by the receiver in response to error free reception of a data frame from the sender. If no *ACK* is received within a timeout period then the sender retransmits the data frame. The sender tries several retransmissions of a data frame before giving up. This way MAC in IEEE 802.11 can shield the wireless link related packet losses from the upper layer protocols.

*viii) **Satellite Transport Protocol***

Similarly to address the TCP's issues over satellite network a new transport protocol termed as Satellite Transport Protocol (STP) was proposed in [HK99B]. Like TCP, STP provides a reliable byte-oriented streaming data service to the applications. The transmitter sends variable length packets to the receiver, storing packets for potential retransmission until the receiver has acknowledged them. However, STP's automatic repeat request (ARQ) mechanism uses selective negative acknowledgements, rather than the positive acknowledgement method of TCP. Packets, not bytes, are numbered sequentially, and the STP sender retransmits only those specific packets that have been explicitly requested by the receiver. Unlike TCP, there is no retransmission timers associated with packets.

From the works discussed above, it can be observed that the proposed solutions address either the congestion control problem of TCP or the problems in TCP due to wireless link. None of these address the two problems together. We however visualize the possibility of tackling these two problems together.

2.2 The Stream Control Transmission Protocol (SCTP)

SCTP [FA04, Ste00, CIALHS03] is a new emerging transport protocol for the Internet. Like TCP, it is a reliable transport protocol. Several new features have been added to it to overcome some of the shortcomings in TCP. SCTP uses SACK based selective acknowledgement technique with TCP like congestion control algorithm. The most important features added in SCTP are multi-homing, multi-streaming and a four way hand shaking mechanism at connection set-up time to protect against the SYN DoS attacks. The multi-streaming feature helps in overcoming the problem of Head-of-Line (HOL) blocking that occurs in TCP as it carries all the data objects in a single stream. With the multihoming feature in SCTP a host can have multiple network addresses and interfaces. There can be more than one path between the same source-destination (sink) pair. The sender maintains a separate set of congestion control parameters for each of the destination addresses it can send to. When needed, SCTP fragments user messages into data chunks to ensure that as a packet is passed to the lower layer it conforms to the path MTU. SCTP data chunks are indivisible units. For purpose of reliability, congestion control and flow control, SCTP assigns each chunk a *transmission sequence number* (TSN). A TSN is unique within it's association until the 32-bit number wraps around and it is independent of the stream on which the chunk is sent. The standard allows SCTP to use delayed acknowledgement while sending back a SACK to the sender after receiving data chunks. With delayed *ACK*, an *ACK* is generated at least every second packet and within 200 ms of the arrival of any unacknowledged Data chunk.

2.2.1 Congestion Control Mechanisms in SCTP

The congestion control mechanism in SCTP is identical to the window-based control scheme of TCP. As in TCP it maintains the control variables *cwnd* and *ssthresh* per destination. Beginning data transmission into a network with unknown conditions or after a sufficiently long idle period requires SCTP to probe the network to determine the available capacity [Ste00]. The slow start algorithm is used for this purpose at the beginning of a transfer, or after repairing a loss detected by the retransmission timer. At the beginning, SCTP initializes *cwnd* for a destination by two packet-size. Every time it receives an *SACK Chunk* it increases the *cwnd* size by either number of bytes *ACKed* or maximum packet size whichever is smaller. If the received *SACK Chunk* does not

advance the cumulative *ACK point* the *cwnd* is not increased. The congestion window is increased in this fashion as long as it is less than or equal to the *ssthresh*. The congestion avoidance phase is started for a path when the size of *cwnd* crosses *ssthresh*. In this phase *cwnd* is increased per RTT by maximum packet size. Upon detection of packet loss through gap reports in SACK chunks SCTP reduces *cwnd* by half. The *ssthresh* is also assigned the same value. SCTP also assumes a packet as lost on the expiry of the retransmission timer. If a packet-loss is detected on timer expiry SCTP initializes the value of *ssthresh* to half of the *cwnd* size and then reduce *cwnd* drastically to one packet-size.

To detect the congestion it depends upon the packet-loss before initiating the recovery process.

2.2.2 Issues in SCTP

In spite of the advanced features provided in the protocol, SCTP's congestion control mechanism is by and large borrowed from TCP. Like TCP, it follows the same strategy for congestion control with the phases of slow-start, congestion avoidance and finally drives the network into congestion. On detection of packet-loss it drastically reduces the *congestion window* size. As a result SCTP inherits most of the congestion related shortcomings of TCP.

In SCTP, due to multi-homing and multi-streaming feature there is possibility of using multiple paths and multiple flows in a connection. However SCTP uses the *alternate path* [Ste00] only for retransmitted chunks and for failover cases with the assumption that the *current path* is likely to be congested. It does not use the alternate path for any other purpose. Recently there have been several proposals [KNKK04, FIOJ05, ISAS03, CGG04] for using the alternate paths to achieve load balancing and higher throughput.

2.2.3 Related Works on SCTP

The SCTP protocol has generated a lot of interest in the research community. Some of the reported works on SCTP are presented below:

i) An Extension for Time Sensitive Traffic

The work in [BF05] proposes extensions to the SCTP protocol for time-sensitive traffic.

a) ***Unreliable time sensitive* extension:** When *retransmissions* reach *reliability level*, the first packet loss (or out of order) detected must trigger a *forward TSN* message. The sender does not wait for multiple loss notifications, allowing for earlier discard notifications of unreliable Application Data Units (ADU). A discard notification is issued after *reliability level* retransmissions only. This behavior allows for a smoother flow of transport streams and application flow(s). Such a discard behavior may cause extemporaneous delivery to the receiver of either an ADU or a *forward TSN* message, none of them having a significant transmission or processing cost.

b) ***Selective forward TSN* extension:** By means of explicit discardable indication (*TSN, StreamId, StreamSeqNum*) in packets following the discardable ADU, the receiver is able to immediately ignore the lost data. Discardable indication is done for ADU transmitted *reliability level* times.

ii) Minimum Delay-based Path Selection

For multi-homed case with path asymmetry, Ribeiro et al.[RL06] proposes minimum delay based path selection technique for SCTP. It selects the one-way path with the lowest delay by taking into account the possible asymmetry of delay values over the forward and reverse paths, under the condition that only one of the available paths is selected for data transmission in each direction. By considering all cross combinations of forward and reverse paths between two multi-homed hosts, this method allows each host to independently determine the lowest delay path and select it for transmitting data to the other host. This allows each one-way data stream to experience the

minimum possible delay, and consequently also minimize the round-trip delay. This approach is targeted for real-time multimedia applications such as voice conversation that require low delay for its periodically transmitted small packets.

iii) Proposal for Modification in cwnd Increment/Decrement

In [IOM04] the authors propose change in SCTP's behavior on *cwnd* increment and decrement during slow-start and congestion avoidance phase for wide area network [IOM04]. During the slow-start phase the *cwnd* is proposed to be increased by either the outstanding data packets acknowledged or the twice of the destination's path MTU, whichever is higher, instead of lower of these two. This is because of the fact firstly that the SCTP sender has to wait for a long time to receive the SACK chunk acknowledging the outstanding data due to high-latency of the network. Secondly, the available rate will be utilized only when the sending rate is high. During the congestion avoidance phase the *cwnd* is increased by (0.01 * *cwnd)* and on the first detection of congestion the *cwnd* is reduced by (0.125 * *cwnd).*

iv) Load Balancing over Multiple Paths

There have been several works on load-balancing over the multiple paths in SCTP. The work in [ASL03] suggested the extension of SCTP for load-sharing, a mechanism to aggregate the bandwidth of all the active transmission paths between the communication end-points. It extends the SCTP, but diverges from it by providing separation between the association congestion control and flow control. For this it requires more meta-data to be added into the packets and also introduces new sequence numbers to maintain per-path ordering information. The Concurrent Multi-path Transfer (CMT) in SCTP proposed in [ISAS03] suggests simultaneous transfer of new data from a source to a destination host via two or more end-to-end paths. The work highlights three negative side-effects of reordering with CMT and proposes the solution for these. The side-effects are unnecessary fast-retransmit at the sender, reduced *cwnd* growth due to fewer *cwnd* updates at the sender and more *ACK* traffic due to fewer delayed *ACK*s. The solution for these are proposed as- a) the sender should infer the lost TSNs using information in SACKs and history information in the retransmission queue, b) the sender should track the earliest outstanding

TSN per destination and update the *cwnd* even in the absence of the cumulative *ACK*s, and c) a CMT receiver should not immediately *ACK* an out-of-order packet but should delay the *ACK*.

The work in [CGG04] proposes bandwidth-aware load balancing over multiple paths in SCTP, with the objective of maximizing the chance of in-order delivery over multiple paths. A rough estimation of the bandwidth available on each round-trip path is performed by sending pairs of SCTP heartbeats on each path and evaluating dispersion of the corresponding heartbeat *ACK*s sent back by the receiver, a well-known technique called packet-pair bandwidth estimation (PPBE). The fact that the return path is unloaded, guarantees that the estimate correctly mirrors the available bandwidth on the forward path. In the current implementation 720-byte heartbeats are sent every 30 seconds. An estimate is made on the earliest time when the opposite end becomes idle on a path after completing reception of last byte of data that is based upon the bandwidth on the path, packet-size to be sent on the path, and one-way propagation time. Then the data-packets are scheduled with a *Fastest Path First* (FPF) approach, which is done by choosing the path for sending the packets that can deliver earliest to the destination.

v) Path-Switching in SCTP

Use of the alternate path for other than retransmission and failover cases has been proposed in [KNKK04], this scheme first calculates application's bandwidth requirement and the available bandwidths in each of the paths. An analysis is then made on which path may fulfill the application's bandwidth requirements. Based on this analysis the path is switched to the alternate path if the primary path cannot fulfill the requirements. In case both the paths do not fulfill the requirements, the path having better bandwidth or lesser delay is used for data transmission. The work [FIOJ05] takes into consideration the application's delay requirement apart from the bandwidth. This comparison between the paths is done periodically to make the switching decision. To probe the networks condition a heartbeat message is sent over each path as per the periodicity.

vi) RTT Based Handover in SCTP

The work in [KMPM04] addresses the handover problem in SCTP in a mobile environment. It uses RTT measurement to take the handover decisions. The scheme periodically measures the RTT of each path and makes a handover decision based on the measurement obtained. It calculates a *Smoothed RTT (SRTT)* using RTT with previous value as baseline. This acts as a low pass filter to minimize the effect of spikes on RTT. The scheme performs a handover to the path with the shortest delay between the two end-points. As the handover is based upon the delay metric rather than the current four timeouts to mark a destination address as inactive, it does not incur any penalty. As a result the scheme leads to fewer retransmissions and higher possibility of seamless handover between networks.

It can be observed that none of the works adequately address the problems in congestion controls inherited by SCTP from TCP. Further, there is ample scope for exploiting the multi-homing and multi-streaming features of SCTP in its congestion control strategy.

2.3 Summary

In this chapter the basic mechanisms of the two reliable transport protocols, namely, TCP and SCTP and their congestion control principles have been reviewed. The main issues with these two protocols are- that they drive the network into deep congestion to the extent of packet-dropping and consider any packet-loss as an indication of congestion. These cause underutilization of the network capacity, high packet latency and high delay jitters. The problem becomes more acute for wireless networks to make these protocols unsuitable to the wireless network where packet-losses due to non-congestion related reasons can be very significant. Another lacuna of SCTP is that it does not use the alternate path in situations other than for retransmissions and failover cases. Thus its multi-homing feature remains largely unutilized for congestion control and load-balancing.

These and some of the related issues have lead to a significant amount of research works. A survey of these works that are relevant to the work of this thesis have

been presented in this chapter. This includes RTT and delay based works in TCP, wireless related works in TCP and some SCTP related works. It can be noticed that there is a need of more work on delay or RTT based congestion control in TCP. The same implies to SCTP too. It has been observed that almost all the schemes proposed for wireless and satellite related issues advocate for major modification in either of the TCP semantics or the Base Station infrastructure. The challenge still remains as to how the congestion and wireless related issues can be resolved in an acceptable manner.

We have also surveyed several works on SCTP that proposes to utilize the multi-homing feature of SCTP. However, these do not adequately address the congestion control and wireless link error aspects. Thus there is need for further study in this area too.

Chapter 3

A New Scheme for RTT Based Congestion Control

In this chapter an analysis is made on Round Trip Time (*RTT*) and its relationship with volume of traffic on the network path. It then considers *RTT* as a measure of level of congestion in the path. This allows us to arrive at a new congestion control scheme based on *RTT*.

3.1 *RTT* as a measure of level of congestion

RTT is the time between the sending of a packet and the arrival of its acknowledgement (*ACK*) at the source. *RTT* depends upon the transmission time as decided by bandwidth/ data rate of the links, propagation delay, processing delay and queuing delay at the routers along the path. In other words it can be written as-

$$RTT = t_{tr} + t_{pd} + t_{pr} + t_{qd} \quad \text{..... (3.1)}$$

where t_{tr} is the transmission time,

t_{pd} is the propagation delay,

t_{pr} is the processing time and

t_{qd} is the queuing delay along the path.

These terms take into account the total time required by the individual components for both the *data* and the *ack* packets over the path. Assuming that the path for the transport connection does not change, once it is set-up, the transmission time t_{tr} and propagation delay t_{pd} for the path will be constant. The third component processing time t_{pr} can also be assumed to be constant for a path. Thus the only component that can vary with time and is dependent on traffic pattern is the queuing delay t_{qd},. It generally increases or decreases gradually with traffic volume. As traffic volume increases in a path the packets get queued up and suffer longer queuing delays.

Fig. 3.1 and Fig. 3.2 give plots of queue length Vs time and RTT Vs time respectively for a path with a bottle-neck link in a ns2 [NS2] simulation run with TCP

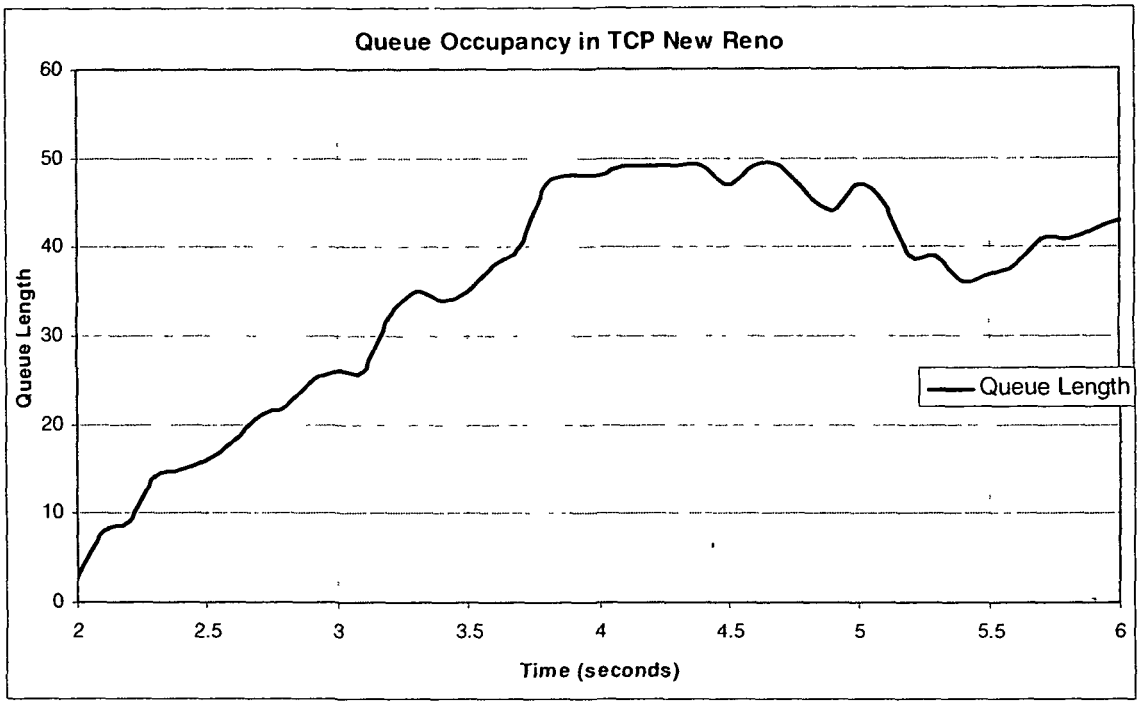

Fig. 3.1: Queue Length (in number of packets) Vs. Time (seconds) in TCP Newreno

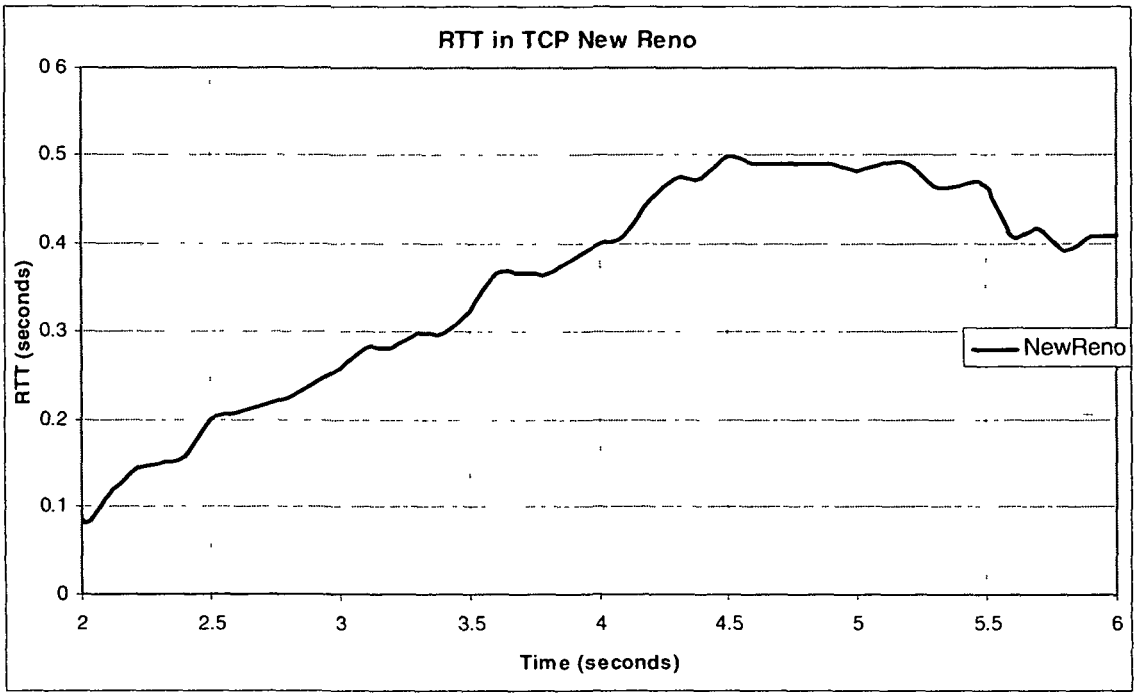

Fig. 3.2: RTT Vs. Time (both in seconds) in TCP Newreno

Newreno. In the experiment when the traffic volume is increased by increasing the number of packets into the network the lengths of the queues in the routers increase. The effect of this on RTT is monitored. The plots display a high correlation between RTT and queue length. An increase in the queue length is always followed by an increase in the RTT value. Similarly a decrease in the queue length is followed by a decrease in RTT. There is however a lag between the change in queue length and the corresponding change in RTT. A close observation also reveals that this lag is also dependent on the queue length. When the queue length increases this lag also increases.

When the traffic volume along the path is low the packets are forwarded almost as soon as they arrive at the router. The queuing delay suffered by the packets is close to nil and the *RTT* value is at its minimum. Thus,

$$RTT_{min} = t_{tr} + t_{pd} + t_{pr} \qquad \text{..... (3.2)}$$

When the traffic along a path increases the queues in the routers along the path also start to grow. This makes the packets to wait longer duration in the routers thus increasing the queuing delay component. As the queue(s) become full the queuing delay reaches a maximum and the *RTT* value also reaches its maximum. In the plot in Figure 3.1 it can be observed that the queue occupancy during the period 4.0-4.5 seconds reaches its highest, which is an indication of highest traffic volume during the period. It can be seen from the Figure 3.2 that RTT also approaches the peak at about the same period. Therefore, the RTT maximum can be expressed as-

$$RTT_{max} = RTT_{min} + t_{qd(full)} \qquad \text{..... (3.3)}$$

where $t_{qd(full)}$ is the queuing delay when the queues along the path are full.

Thus, depending upon the traffic load the *RTT* value will vary between RTT_{min} and RTT_{max} with varying queue occupancy in the routers. The *RTT* value can therefore be used as a measure for the traffic volume in the network.

3.2 Use of RTT as a Control Parameter for Congestion Control Avoiding Packet Dropping

From the discussion in the previous section it is clear that the *RTT* value varies between RTT_{min} and RTT_{max}, and it reaches RTT_{max} when the queue length attains the maximum with the queue becoming full. When the queue in a router becomes full it starts dropping packets (with the assumption that queuing mechanism used in the bottle-neck link is DropTail or FIFO). Thus packet losses start to occur when *RTT* value reaches a maximum. To avoid packet loss it is necessary to avoid high queue occupancy. This queue occupancy is reflected in the value of *RTT*. Therefore, if the *RTT* value can be controlled to remain within a range below RTT_{max}, the queues in the routers along the path will remain partially filled and packet dropping will not occur. Also if the *RTT* value is controlled to hover around a threshold value $RTT_{threshold}$ that lies within the range (RTT_{min}, RTT_{max}), it will be possible to restrict the delays suffered by packets to within desired limits by choosing the value of $RTT_{threshold}$ appropriately. This will ensure that the queue length in the routers is also controlled within the desired limit. The $RTT_{threshold}$ value can also be fixed sufficiently above the lowest end of the range so that the queues in the routers have sufficient supply of packets to feed the output links. This will result in an optimal throughput without suffering packet losses.

One important aspect that has to be kept in mind is fairness. The packet injection rate of the flows should be such that each of them gets a fair share of the bandwidth. Fairness may suffer if a flow sets its *congestion window* size higher than appropriate. That will push *RTT* high which in turn may force the other flows to reduce their *congestion window* thus bringing down their share of the bandwidth. This will lead to drop in fairness.

Based on this reasoning a scheme has been devised for restricting the rate of injection of packets into the network to make the *RTT* value hover around a $RTT_{threshold}$ value. The scheme is discussed in the following section.

3.3 The Proposed Scheme

The idea is to control the injection of packets into network at such a rate so that the queue occupancy does not exceed a limit so as to push the *RTT* value beyond a limit. Similarly, care needs to be taken to maintain the queue occupancy such that the throughput does not fall below the desired level. For this the packet injection rate is maintained such that the *RTT* value does not touch RTT_{min}. The way to control the packet injection rate is by adjusting the *Congestion Window* size at the source. Since it is desired to keep the *RTT* value hovering around a threshold $RTT_{threshold}$ one possible way to achieve this is whenever *RTT* exceeds $RTT_{threshold}$ the *Congestion Window* size is reduced in proportion to the *RTT* value exceeding $RTT_{threshold}$. The reverse can be done when the *RTT* value falls below $RTT_{threshold}$.

Thus the scheme basically involves monitoring of the *RTT* of the packets transmitted by a sender and increasing or decreasing the rate of injection of packets into the network by appropriately adjusting the *Congestion Window (cwnd)* size depending on *RTT* falling below the $RTT_{threshold}$ value or its going above it. To decide on the increment and decrement required in the *cwnd* size, the equation (3.4) below is used.

$$cwndIncr = \frac{(RTT_{threshold} - RTT)}{min\,(RTT, RTT_{threshold})} \times Segment\ Size \qquad \text{--- (3.4)}$$

The *cwnd* is altered as-

$$cwnd = cwnd + cwndIncr \qquad \text{--- (3.5)}$$

The *cwndIncr* will be positive when *RTT* is lower than $RTT_{threshold}$ and will be negative when *RTT* is higher. The *cwndIncr* function is devised in such a way that when the difference between $RTT_{threshold}$ and *RTT* is large the *cwndIncr* value is also large so that the *cwnd* is increased/ decreased sufficiently to allow the *RTT* to quickly catch-up with the $RTT_{threshold}$ value. The *RTT* value is monitored continuously for every *ACK* that arrives and *cwnd* is updated accordingly.

Once again the plots for queue occupancy and corresponding RTT have been obtained as shown in Fig. 3.3 and Fig. 3.4 by applying the technique above in TCP

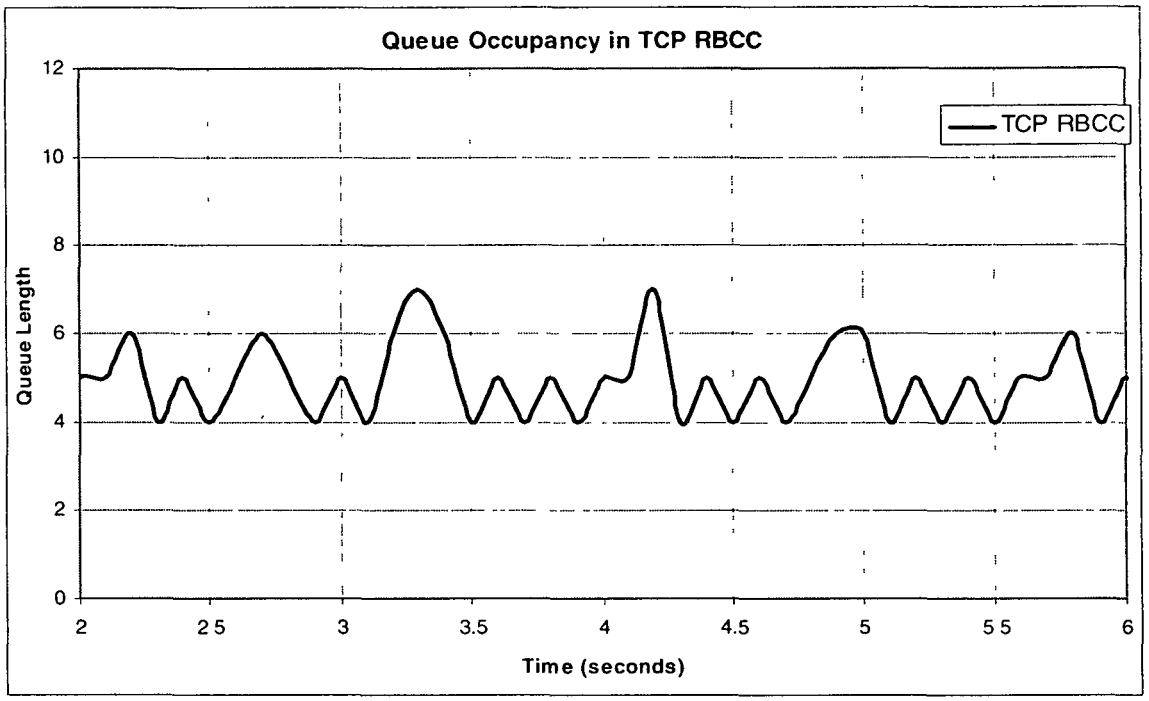

Fig. 3.3: Queue Length (number of packets) Vs. Time(seconds) in TCP RBCC

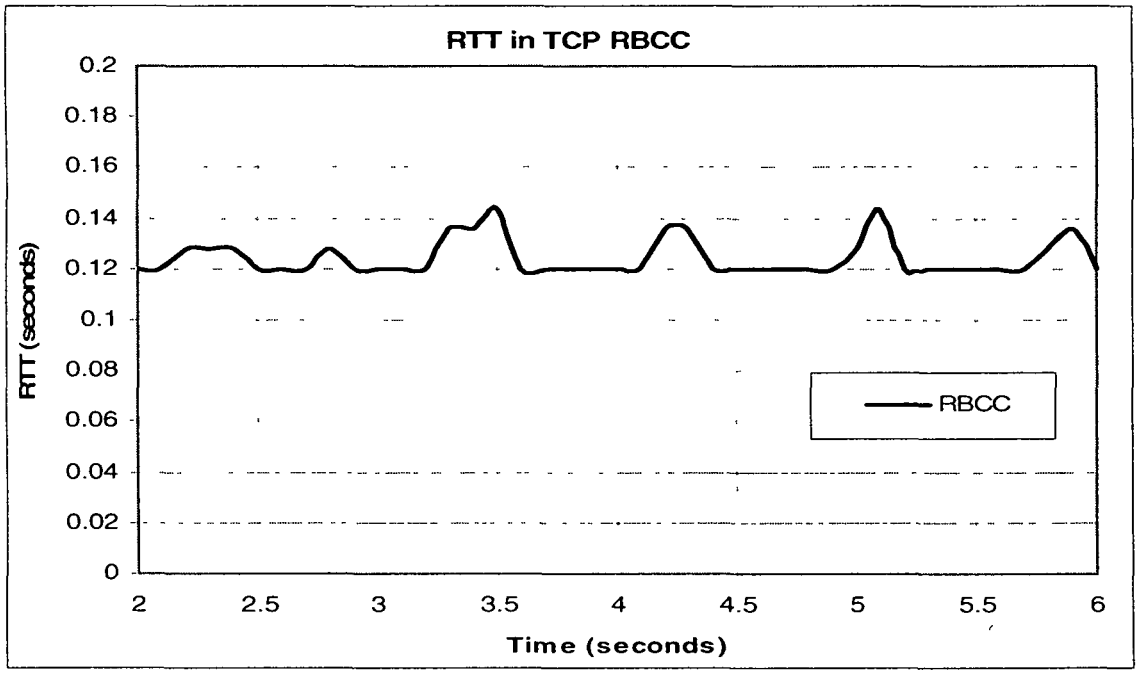

Fig. 3.4: RTT (Latency) Vs. Time (both in seconds) in TCP RBCC

Newreno for adjustment of the *Congestion Window*. This resulting congestion control technique for TCP shall be referred to as *RTT Based Congestion Control (RBCC)* or *TCP RBCC*. In Figure 3.3, it can be seen that TCP RBCC maintains low queue occupancy by anticipating congestion before-hand and controls the packet injection rate

to achieve this. As a result of this, it can be seen in Figure 3.4 that TCP RBCC is able to maintain low RTT or equivalently low packet latency by many folds than what is possible with TCP Newreno (Figure 3.2). This low latency results from the low queue length at the routers due to the controlled injection rate at the source.

However, it is also necessary to keep in mind the resulting *throughput* for the flow. The queue occupancy in the routers should not be brought down to the level where the throughput goes below the desired level. Therefore it is necessary to fix the $RTT_{threshold}$ value appropriately. In fact there is a tread-off between *throughput* and *packet latency*. In this tread-off $RTT_{threshold}$ can be used as the control parameter in the proposed congestion control scheme. The ways to fix the $RTT_{threshold}$ are discussed in the following chapter.

3.4 Summary

An analysis has been made on the relationship between traffic volume and *RTT*. It is shown how *RTT* varies according to the traffic volume as reflected by the queue length in the routers. Thus it is argued that *RTT* indicates the state of congestion in a path and therefore can be used as a measure of level of congestion. Based on this a new scheme has been proposed which monitors the *RTT* value and adjusts the *Congestion Window* to maintain *RTT* value hovering around a threshold value decided upon as per requirements such as desired throughput and packet latency. The RTT measurement is fine-grained and also the probing is done for each packet using the time-stamp option in TCP.

Chapter 4

Employing RBCC in TCP for Wired Network

In this chapter, the proposed new RTT based scheme for congestion control in TCP as used in wired networks is presented. Schemes for fixing the *threshold RTT* statically and dynamically are also presented in this chapter. Finally the results of simulation-based experiments in wired network environment comparing the performance of the new scheme with TCP NewReno, Vegas and SACK are presented.

4.1 TCP RBCC for Wired Network

The congestion control in TCP with the new scheme RBCC in a wired network involves the following:

i) Initialize the size of *cwnd* to one segment at the time of connection setup.

ii) Compute fine-grained *RTT* using TCP Timestamp option for each packet on the reception of it's ACK packet.

iii) Maintain a *threshold RTT* value computed statically or dynamically as discussed in Section 4.3 and Section 4.4.

iv) Change the size of *cwnd* based on the equation (3.4).

v) For retransmission time-out estimation follow TCP NewReno.

vi) Use fast retransmit, fast recovery and recovery in case of more than one packet loss in a single-window as in TCP NewReno.

4.2 Addressing TCP's Congestion Control Problem

Employing the proposed mechanism, as described in the previous section, TCP RBCC no longer needs to wait for occurrence of an extreme congestion situation leading to packet loss for detection of congestion. As can be noted from the

observations made in the previous chapter, RTT clearly reflects the behavior of bottle-neck queue and if *cwnd* is controlled based on RTT then queue-occupancy is also controlled in the routers. Therefore the congestion can be controlled independent of the packet losses. Neither is the subnet required to push the traffic volume to the extreme of buffer overrun. In this way the routers are able to maintain low queue length so that the end-points observe lower latencies. This new technique, having the capability of reducing the congestion related packet losses, can lower the retransmitted traffic volume. Thus the subnet is able to transmit more meaningful packets and can have better utilization of the scarce network resources rather than overloading the network with retransmitted packets.

4.3 Setting Threshold RTT Statically

In the absence of a dynamic mechanism for computation of the value for $RTT_{threshold}$, it can be set statically based on a priori estimation of the RTT_{min} as per equation (3.2). This needs knowledge of the characteristics of the links along the path between the source and the destination hosts. The $RTT_{threshold}$ value can be taken as a product of a constant factor $F\ (>1)$ and RTT_{min}, that is:

$$RTT_{threshold} = F \text{ X } RTT_{min} \qquad \text{------- } (4.1)$$

The value of F will decide the allowed queuing delay suffered by the packets and that in turn will decide the allowed length of the queues in the routers.

4.4 Algorithm for Dynamic Computation of Threshold RTT

Setting Threshold RTT ($RTT_{threshold}$) statically may not always be possible as the characteristics of the path may not. at times, be known. Ability to decide the value of the threshold on-the-fly and dynamically makes the congestion control scheme usable in networks whose characteristics are unknown. An algorithm has therefore been devised for computing the $RTT_{threshold}$ dynamically. In this the computation is started after the connection is established. It is based on probing that lasts for a finite number of iterations after which $RTT_{threshold}$ gets stabilized, except on certain rare conditions. Figure 4.1 depicts the state transition diagram for the algorithm.

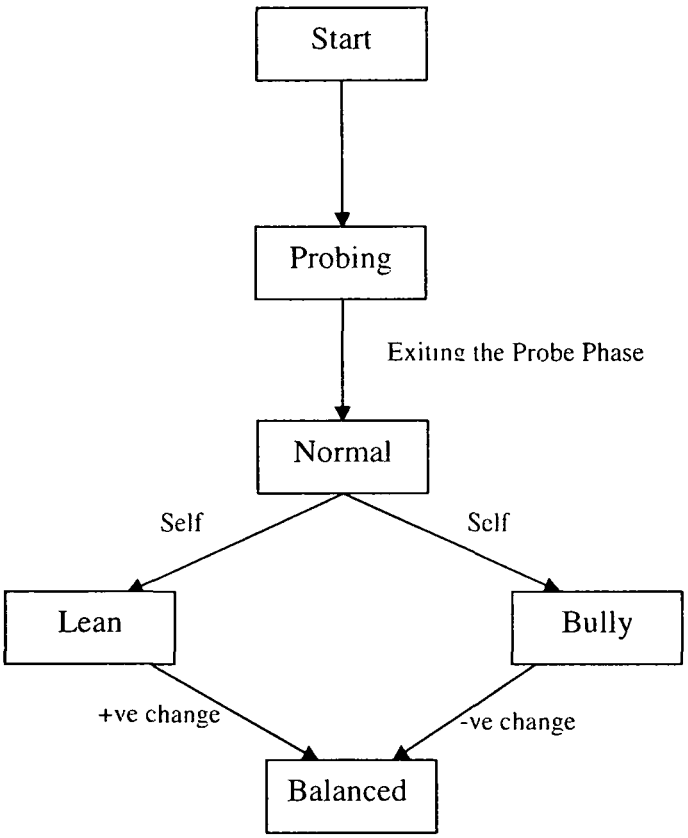

Fig. 4.1: State Transition Diagram of Algorithm for the Dynamic Computation of Threshold RTT.

The algorithm consists of five heuristics for altering the value of $RTT_{threshold}$ in two broad scenarios viz. probing phase and normal phase. The entire set of heuristics is based on two intuitive elements, namely variation in *cwnd* and variation in *RTT*. With the use of equation (3.4), if value of $RTT_{threshold}$ is changed then the same is expected to be reflected in the value of *cwnd* and *RTT*. So the heuristics take cautious steps of changing the value of $RTT_{threshold}$ and observing variations in *cwnd* and *RTT*. Though all the five heuristics observe the variations in *cwnd* and *RTT*, the difference among them is the amount and the sign of the variation. The $RTT_{threshold}$ is increased when *RTT* is not rising and *cwnd* is also not growing. Similarly, it is decreased when *RTT* is rising and *cwnd is* also growing. Below, in part A we define the terms used, in part B we present the five heuristics, and in part C the steps involved in the algorithm are described.

A. Definitions:

thresh_rtt : Threshold *RTT*

Bully : A state in which the value of *thresh_rtt* is too high and *cwnd* passes through a phase of high growth.

Lean : A state in which the value of *thresh_rtt* is too low and *cwnd* passes through a phase of negative or no growth.

rtt_count : A counter to keep track of consecutive growth or fall in *RTT*. It is set to zero when there is no change in *RTT*. Incremented if *RTT* is falling and decremented when *RTT* is rising.

incr_clamp : A threshold value for *rtt_count*. It is set to decide if the *thresh_rtt* value needs to be enhanced.

decr_clamp : A threshold value for *rtt_count*. It is set to decide if the *thresh_rtt* value needs to be reduced.

A Probe : A *probe* is a sampling of the connection's parameters, namely, *RTT*, *thresh_rtt*, and *cwnd* for making a decision on updating the value of *thresh_rtt*.

Probe_clamp: A threshold for the number of probes to be performed before exiting from the probe phase.

pp_increment: It is a positive change (in %) to be made on the *thresh_rtt* on a probe.

pn_decrement: It is a negative change (in %) to be made on the *thresh_rtt* on a probe.

lp_increment : It is a positive change (in %) to be made on the *thresh_rtt* during a *Lean* state.

bn_decrement: It is a negative change (in %) to be made on the *thresh_rtt* during a *Bully* state.

B. Heuristics

I. Heuristic for Positive Increment on Probe

The *threshold RTT* needs incrementing under the following conditions:

1. When the $RTT_{threshold}$ is lower than *RTT*, as per equation (3.4) *cwnd* will not grow but may fall. If this continues for several consecutive instances it will mean that $RTT_{threshold}$ is set at a low value than desired. The same may be the

case if *RTT* remains unchanged or drops for several consecutive ACKs. In such situations $RTT_{threshold}$ needs incrementing.

2. At the beginning of setting-up of a TCP connection when there is no congestion and the *RTT* has not been growing consecutively for several instances.

Thus, if *cwnd* has not grown continuously for γ_1 ($2 \leq \gamma_1 \leq 3$) occasions, *rtt_count* ≥ 0 and *cwnd* ≤ 1.5, or if *rtt_count* $\geq$ *incr_clamp* on alternate occasions then apply an increment of -

$$pp\_increment = \alpha_1 / \textit{No. of increments in all the probes}$$

where $7 \leq \alpha_1 \leq 15$ is a constant,

Choosing the value of the constant α_1 above 15 would make $RTT_{threshold}$ to rise aggressively and can create unfairness among the competing flows. On the other hand, a choice of α_1 lower than 7 would be too conservative and may end up having too low a value for $RTT_{threshold}$ than desired. The reason for dividing α_1 by *number of increments in all the probes* is to prevent the aggressiveness in the growth of $RTT_{threshold}$ which ensures that amount of *pp_increment* gets reduced as the number of probes increases. This also helps in maintaining fairness. Similarly, choosing $\gamma_1 < 2$ will make it aggressive and $\gamma_1 > 3$ will make it conservative.

Here, the *rtt_count* requires to be positive indicating that either *RTT* is falling or not changing and there is still room for $RTT_{threshold}$ to grow. However, at the start of a connection for a few probes this condition is relaxed by setting the value of *incr_clamp* to -5. The *incr_clamp* is increased on each probe until it reaches value 2. This helps $RTT_{threshold}$ for a flow to grow sufficiently. Otherwise it may be forced to remain low in the presence of already established flows.

II. Heuristic for Negative Change on Probe

The $RTT_{threshold}$ needs decrementing under the following condition:

- When *cwnd* has been growing and also *RTT* has been rising for consecutive instances.

Thus, if *cwnd* grows for γ_2 (e.g. 2) consecutive occasions, and *rtt_count* ≤ *decr_clamp* then negative incentive is applied as:

pn_decrement = β_1 / *No. of decrements in all the probes*

where $0 < \beta_1 < \alpha_1$, normally β_1 should lie between 3 to 8.

γ_2 is positive and normally *lies in the range* of 2 to 3.

Choosing value of β_1 below 3 will make the rate of reduction of $RTT_{threshold}$ too slow delaying the convergence. On the other hand choosing a value above 8 may lead to $RTT_{threshold}$ to be reduced too quickly and may end up having a too small a value for $RTT_{threshold}$ than desired. The reason for dividing β_1 by *number of decrements in all the probes* is to prevent the aggressiveness in the shrinking of $RTT_{threshold}$ This also helps in maintaining fairness. Similarly choosing value lesser than 2 for γ_2 would make it aggressive and having larger than 3 would make it conservative in the shrinking of $RTT_{threshold}$.

Here the sign of *rtt_count* is required to be negative indicating that *RTT* is rising. Initially, this condition is relaxed by setting the value of *decr_clamp* to -10. The *decr_clamp* is then increased gradually on each iteration until it reaches -2. This prevents starting the reduction in the value of $RTT_{threshold}$ too quickly at the beginning of a TCP connection.

III. Heuristic for Ending Probe

The probe has to be ended on one of the following conditions:

1. When the *cwnd* has achieved a reasonable growth and the queue occupancy for the flow has also risen a bit. This ascertains that the threshold has passed through some growths.
2. When the flow has undergone a threshold number of probes including the positive and negative changes.

Thus. if *cwnd* is grows continuously for γ_{31} (e.g. 5) occasions, and *rtt_count* $\leq \gamma_{32}$ (e.g. -5), or if the number of probes has crossed the *probe_clamp* then stop the probing and enter into the normal state.

Here, γ_{31} is positive and normally lies in the range of 4 to 7,

γ_{32} is negative and normally lies in the range of -4 to -7

Experiments show a value of 20 for *probe_clamp* to be sufficient. Too low a value may end the probe pre-maturely and too high a value may prolong the probe unnecessarily. Choosing the values of γ_{31} and γ_{32} outside the specified ranges may either end the probe too quickly or prolong the probe.

IV. Heuristic for lean state

A flow is declared *lean* on the following condition:

- The *cwnd* is passing through a phase of very low value causing the traffic volume to be very low.

Thus, if *cwnd* is not growing continuously for γ_{41} (e.g. 15) occasions, *rtt_count* ≥ 0 *cwnd* ≤ 2, then determine if this is happening repeatedly for next γ_{42} (e.g. 15) occasions, then declare the state as *Lean* and apply the following positive incentive:

lp_increment = α_2 / *No. of last back-to-back lean states*

where α_2 is positive lies in the range of 3 to 6.

γ_{41} is positive and normally lies in the range of 10 to 20,

γ_{42} is positive and normally lies in the range of 10 to 20

Choosing value of α_2 above 6 would make $RTT_{threshold}$ to rise aggressively and can create unfairness among the competing flows. Similarly, the choosing a α_2 lower than 3 would be too conservative and may end up having too little value for $RTT_{threshold}$ than desired. The reason for dividing α_2 by *number of last back-to-back lean states* is to prevent the aggressiveness in the growth of $RTT_{threshold}$. This also helps in maintaining fairness. Similarly, choosing too little value for γ_{41} and γ_{42} would make it to declare the state as *lean* prematurely and having too much would

unnecessarily delay the declaration. The first condition also may bring some instability and the second one may starve the flow for too long.

V. Heuristic for bully state

A flow is declared *bully* on following condition:

- The *cwnd* grows substantially and causes a high level of congestion.

Thus, if the *cwnd* grows continuously for γ_{51} (e.g. 15) occasions, *rtt_count* ≤ *decr_clamp* and if the value of *cwnd* is more than 2, then determine if this is happening repeatedly for next γ_{52} (e.g. 15) occasions then declare the state as *bully* and apply the following negative incentive:

$$bn\_decrement = \beta_2 / No.\ of\ last\ back\text{-}to\text{-}back\ Bully\ states$$

where $0 < \beta_2 < \alpha_2$, β_2 normally lies in the range of 2 to 5,

γ_{51} is positive and normally lies in the range of 10 to 20,

γ_{52} is positive and normally lies in the range of 10 to 20.

Choosing value of β_2 below 2 would make $RTT_{threshold}$ to be reduced very slowly and it may take too long to reach an appropriate threshold value. Choosing a value above 5, on the other hand, would cause fast reduction in $RTT_{threshold}$ and may end up having too small a value for $RTT_{threshold}$. The reason for dividing β_2 by the *number of last back-to-back bully states* is to prevent aggressiveness in the shrinking of $RTT_{threshold}$, this also helps in minimizing chances of starving the connection or reaching *Lean* state. Similarly choosing too small value for γ_{51} and γ_{52} would make it to declare the state as *Bully* prematurely and having too large value would make it too late in declaring the state.

C. Steps for Dynamic Computation of Threshold RTT

1. Initially *thresh_rtt* ($RTT_{threshold}$) is assigned a value 10% above the presently available *minimum RTT*.
2. Start the *Probe phase* and repeatedly check for *Heuristic I, Heuristic II* and *Heuristic III.*
   a. If *Heuristic I* is successful apply *pp_increment* to *thresh_rtt.*
   b. If *Heuristic II* is successful then apply *pn_ decrement* to *thresh_rtt.*

c. If *Heuristic III* is successful then exit from the *Probe phase*.

3. Enter the *Normal Phase* and reset the values of *incr_clamp* and *decr_clamp* to 10 and -10 respectively. Continue to check for *Heuristic IV* and *Heuristic V* each time a new *RTT* and *cwnd* are calculated after receiving a fresh *ACK*.
   a. If *Heuristic IV* is successful then declare the state as *Lean* and increase *thresh_rtt* by *lp_increment*.
   b. Otherwise if *Heuristic V* is successful then declare the state as *Bully* and decrease *thresh_rtt* by *bn_ decrement*.
   c. In steps a and b above reset the counters for *cwnd* and *RTT* growth (*rtt_count*).
4. While applying negative changes in steps 2.b and 3.b make sure that the minimum value of *thresh_rtt* is 2.5% above the presently available *minimum RTT*.

4.5 Experimental Study on Performance of RBCC in Wired Network Environment

In this section we present the results of simulated experimental studies on the performance of RBCC in a wired network environment. First, we study the behavior of RBCC in respect of packet loss, throughput, fairness, packet latency, and queue occupancy with varying $RTT_{threshold}$ values. Then we compare the performance of RBCC with NewReno, Vegas and SACK in respect of packet loss, throughput and fairness for varying traffic load (simulated by varying the number of connection) and the packet latency and queue occupancy patterns of RBCC with the existing schemes for fixed number of connections. Finally, we compare the performances of RBCC with fixing of the $RTT_{threshold}$ statically and dynamically.

In these experiments, the packet-loss is measured with the number of bytes retransmitted at the sender. Throughput is measured as number of bytes transmitted minus number of bytes retransmitted. When there are more than one flow, the packet-loss and throughput computed are in aggregate. The queue-occupancy is measured using length of queue in the bottle-neck link where packets are queued waiting for transmission. The latency is measured using RTT itself. While measuring fairness the equation-(1.1) in Chapter 1 is used.

4.5.1 Topology and Environment for Experiments

The experiments have been carried out using the Network Simulator (NS2) package [NS2] on Linux platform. The topology shown in Fig.4.2 was used for the experiments. Here every source-sink node pair ($Sourc_i$ - $Sink_i$, *i=0 to 4*) is connected through a common 1 Mbps bottleneck link (R_1-R_2) with 10 ms propagation delay. Each of the feeding links has also a bandwidth of 1 Mbps and 10 ms propagation delay. The data transfers are done by running *ftp* on a very large file at every source node. The queuing discipline used is Drop-Tail (i.e. FIFO Queuing). The queue buffer limit at the router is set to 50.

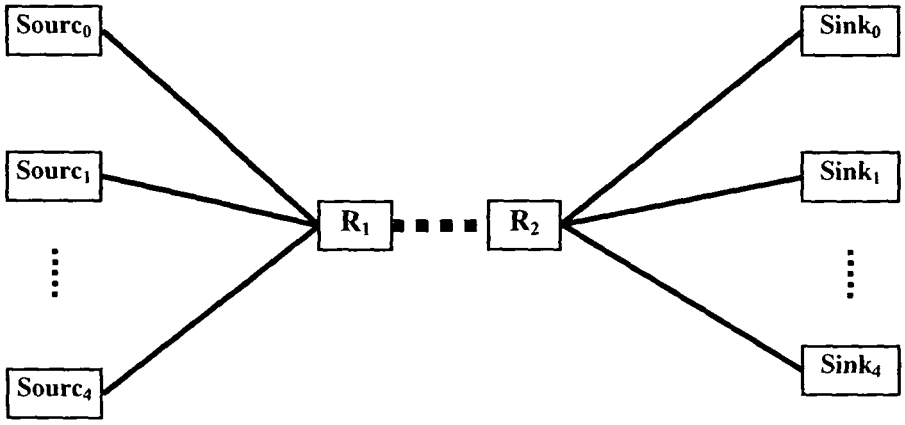

Fig. 4.2 Topology for Experiments in Wired Network

To increase the traffic load, simultaneous connections are setup between each of the source-sink pairs with equal distribution. For example, if the total number of flows is 10, then 2 connections are setup between each of the source-sink pairs.

TCP window size is set to 20, three number of duplicate ACKs are used for Fast Retransmit, TCP clock granularity is set to 0.01 seconds, the slow start threshold is set to 20 and immediate ACK is performed as per RFC [APS99]. These parameters are common across all the variants of TCP (including the new scheme). For TCP Newreno (and also for RBCC), slow-but-steady variant is used as per RFC [FH99] with retransmit timer reset after each partial new ACK. Also, *cwnd* is set to *ssthresh* upon leaving fast recovery and upon partial ACK (i.e. no window deflation option) [FH99].

For TCP Vegas the values of *Alpha, Beta and Gamma* parameters are set to 1, 3 and 1 respectively, which are the default values. For TCP SACK, maximum SACK blocks is set to 3.

4.5.2 Ability of the Scheme to Control Packet-loss

The first experiment was to verify the ability of the scheme to control packet loss. Five *ftp* connections on top of TCP RBCC are set up between each of the source-sink pairs and data injection rate is varied by varying the $RTT_{threshold}$ value. Each of the instances of the simulation runs for 100 seconds. It is observed that the packet-losses could be avoided by limiting $RTT_{threshold}$. Packet-loss occurrence starts only when the $RTT_{threshold}$ value is set above a certain limit. Fig.4.3 shows a plot of packet-loss volume against $RTT_{threshold}$ (represented by F). In the experiment the packet losses start for F value of about 5.0.

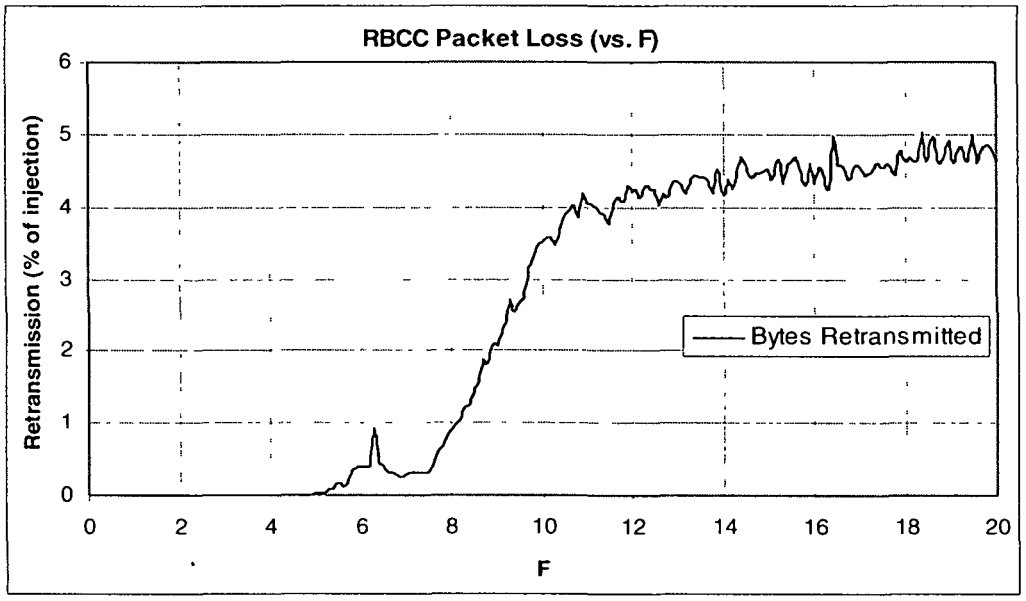

Fig. 4.3 Packet Loss Vs. $RTT_{threshold}$ Value in TCP RBCC

4.5.3 Throughput Sensitivity to $RTT_{threshold}$

Experiments have been conducted to examine the sensitivity of the throughput to the $RTT_{threshold}$ value. Five *ftp* applications have been run continuously for 100 seconds between the same pairs of source nodes and the sink nodes on top of the TCP RBCC connections. With the bottleneck link between R_1 and R_2 limiting the data rate to 1

Mbps the maximum possible volume of data transfer over the period is 100 M bits. Fig. 4.4 shows a plot of *Throughput Vs. F*. It can be seen that as the $RTT_{threshold}$ value is increased by increasing F, the throughput (in % of capacity) quickly attains its maximum possible value and it remains at that level for a wide range of $RTT_{threshold}$ values before the throughput starts falling when the packet losses start. It appears that the throughput is not sensitive to $RTT_{threshold}$ over a wide range implying that fixing of the $RTT_{threshold}$ value is not too critical as far as throughput is concerned.

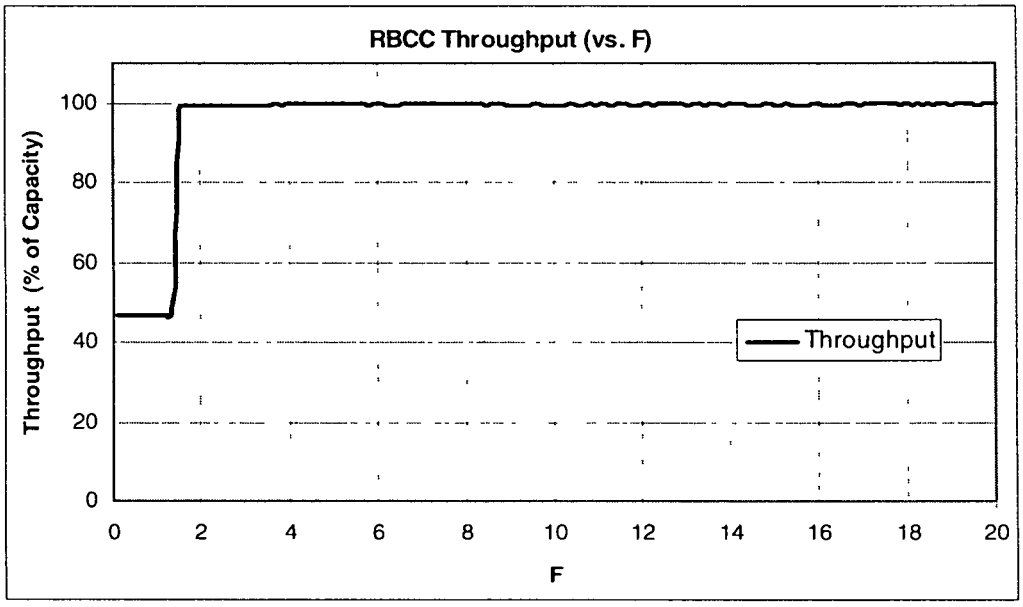

Fig. 4.4 Throughput Vs. $RTT_{threshold}$ Value in TCP RBCC

4.5.4 Ability of the Scheme to Remain Fair

The fairness of the scheme was tested against varying $RTT_{threshold}$ (F) and with a fixed number (5) of Connections. It was found that RBCC maintains fairness above 80% up to an F value of 6 (as shown in Fig. 4.5). Beyond this the fairness drops a bit, but always remains above 60%.

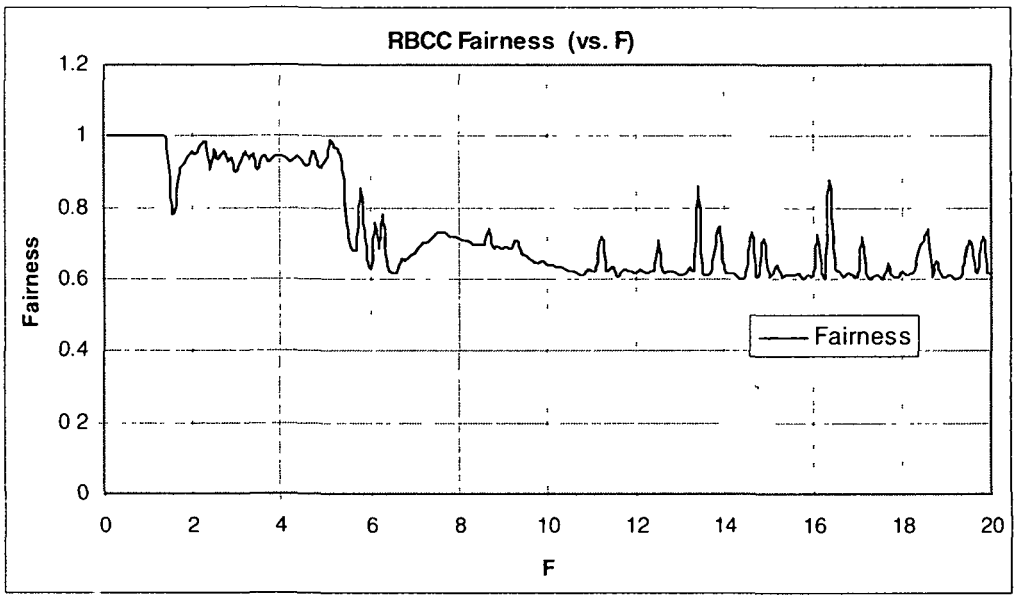

Fig. 4.5 Fairness Vs. $RTT_{threshold}$ Value in TCP RBCC

4.5.5 Ability of the Scheme to Control Packet-Latency

Fig. 4.6 shows the latency suffered by packets in the new scheme for different values of $RTT_{threshold}$ as fixed with different of F. From the figure it can be observed that the packet latency and the delay jitter can be limited by choosing appropriate value for $RTT_{threshold}$. Latencies can be lowered by choosing lower value for $RTT_{threshold}$. Thus packet latency can be controlled as desired.

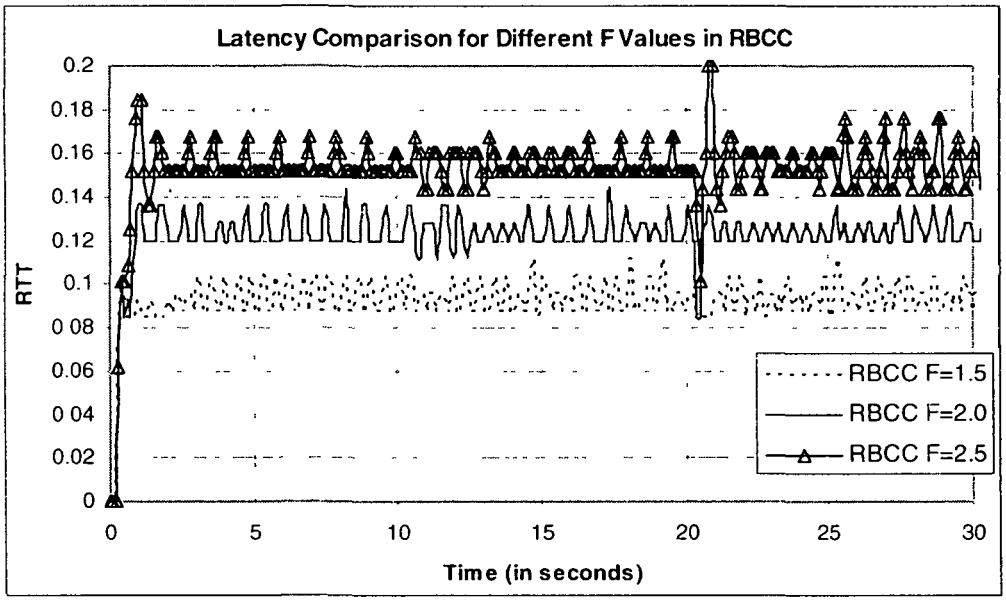

Fig. 4.6 RTT Vs. Time for different values of F (No. of Connections =5) in TCP RBCC

4.5.6 Ability of the scheme to control bottle-neck queue-occupancy

Fig. 4.7 shows plots for queue-length in the bottle-neck link for each case of the three different values viz. 1.5, 2.0 and 2.5 of F in the new scheme. As can be seen, F=1.5 is maintaining lower queue-occupancy than the F=2.0 and 2.5; and F=2.0 is having lower than 2.5. From this it can be observed that the queue-occupancy also can be limited by choosing appropriate value for $RTT_{threshold}$.

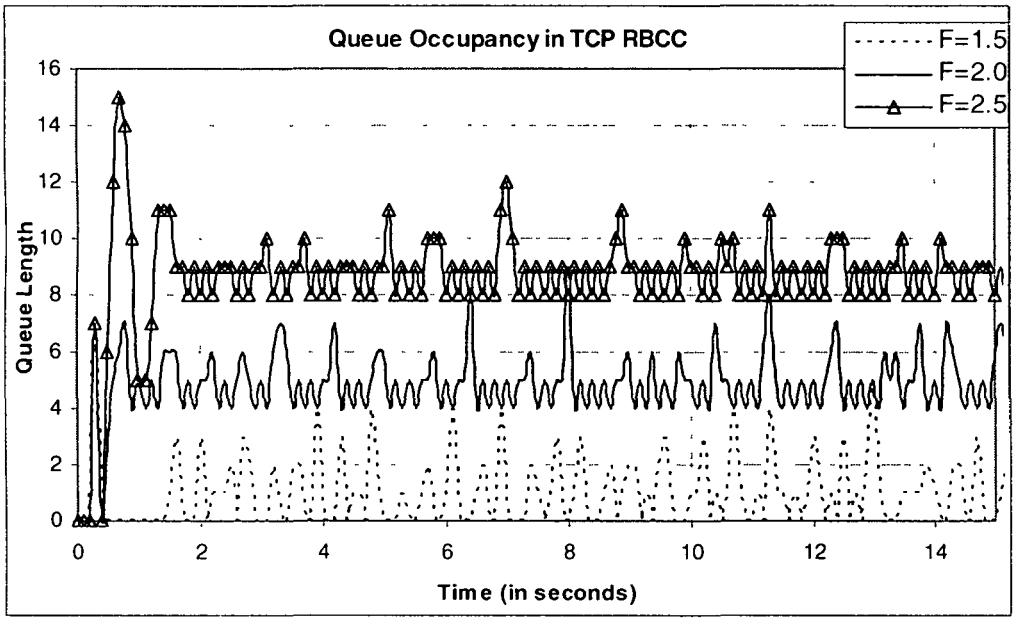

Fig. 4.7 Queue-Occupancy Vs. Time for different values of F (No. of Connections =5) in TCP RBCC

4.5.7 Comparing Performance of TCP RBCC with NewReno, Vegas and SACK

A set of experiments have been carried out to compare the performance of the new scheme with that of TCP NewReno, Vegas and SACK. We observe the performance of these schemes in terms of packet loss, throughput and fairness against varying traffic load. The variation in the traffic load is achieved by varying the number of simultaneous connections through the network. We also study the performance in terms of Packet latency, delay jitter and queue occupancy. Here we measure the performance for a fixed number of connections. The $RTT_{threshold}$ set for RBCC for measure of all the performance parameters was with $F=2$.

4.5.7.1 Packet Loss

Fig. 4.8 shows the plots of packet losses with increasing number of connection for the four schemes as percentage of the packets injected in to the network. In this experiment the packet losses were estimated in terms of retransmitted packets. It can be observed that the packet loss is negligible for RBCC even for very high traffic load. For TCP Vegas the packet is very low for low load. But, after the load crosses a limit its packet loss grows very fast. For NewReno and SACK packet losses are significant even at low load and continue to grow with load.

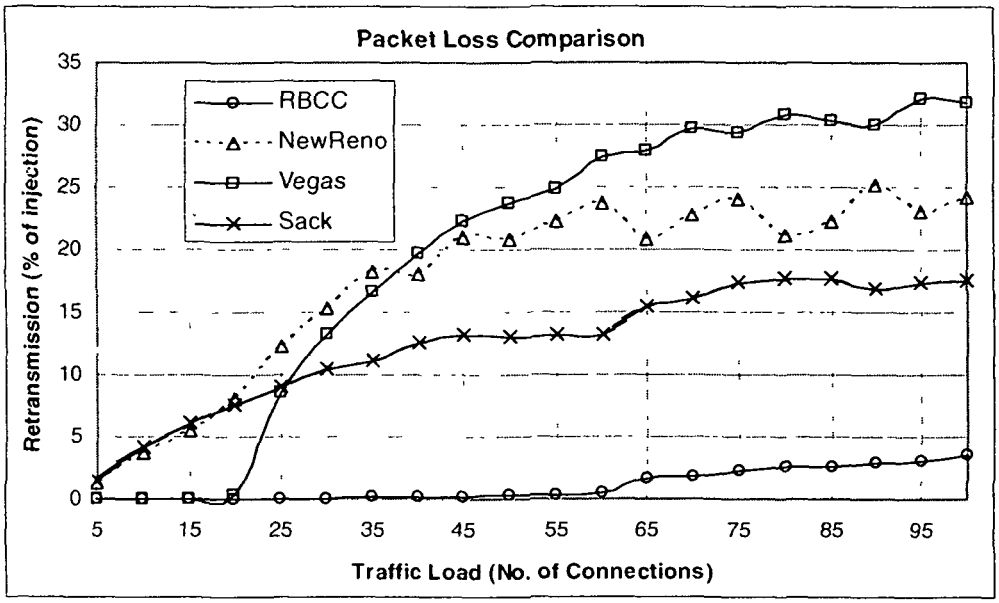

Fig 4.8 Packet Loss Vs. Traffic Load in TCP RBCC (F=2.0), NewReno, Vegas and Sack

4.5.7.2 Throughput

Fig. 4.9 plots the throughput for the four schemes as percentage of the capacity of the network against increasing traffic load. The new scheme, RBCC, is able to maintain a throughput of near 100% even a very high load. For Vegas, the throughput is near 100% at low load and starts dropping slowly with load. SACK maintains a throughput of about 95%. For NewReno the throughput drops from little over 95% at low load to below 90% at high load.

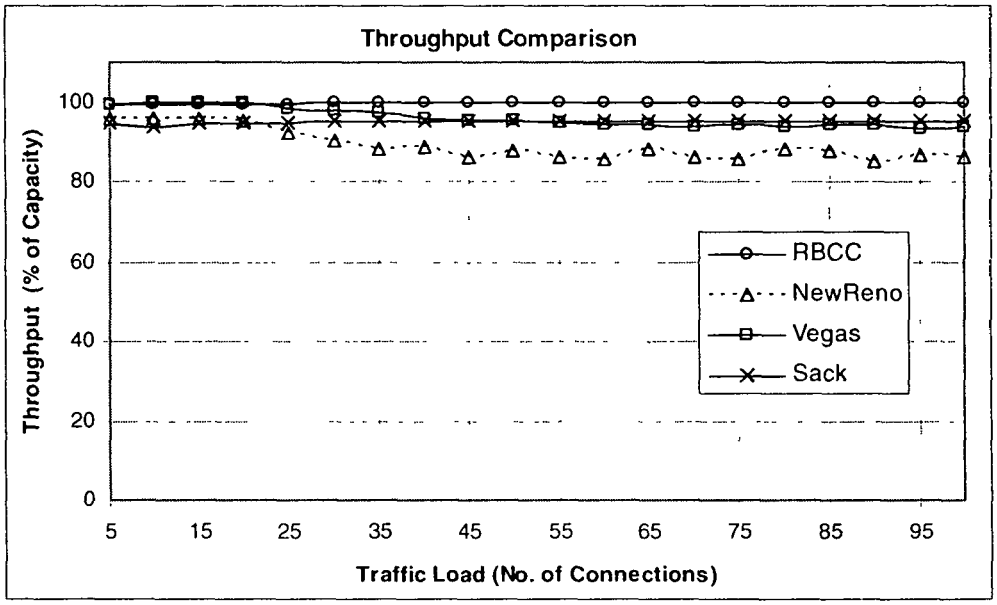

Fig. 4.9 Throughput Vs. Traffic Load in TCP RBCC (F=2.0), NewReno, Vegas and Sack

4.5.7.3 Fairness

Fig. 4.10 shows the plots of fairness of the four schemes against increasing traffic load. It was observed that RBCC maintains a near 100% fairness for a range of traffic load and remains fairer than that of NewReno, Vegas and SACK most of the time. At very high traffic load its fairness starts dropping slowly but continues to perform better than NewReno and SACK. The fairness for the other three schemes fluctuates with varying load.

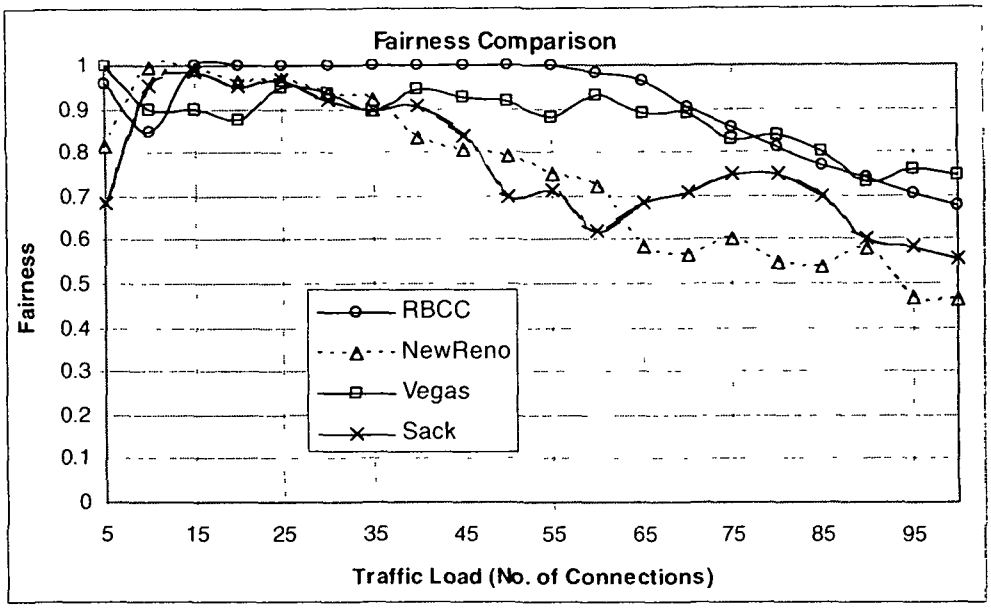

Fig. 4.10 Fairness Vs. Traffic Load in TCP RBCC (F=2.0), NewReno, Vegas and Sack

4.5.7.4 Packet Latency

Fig 4.11 shows the plots of packet latency and delay jitter suffered by packets in the new scheme, RBCC in comparison to those of TCP Newreno, Vegas and SACK. Here the experiments were conducted with 5 simultaneous TCP connections in the

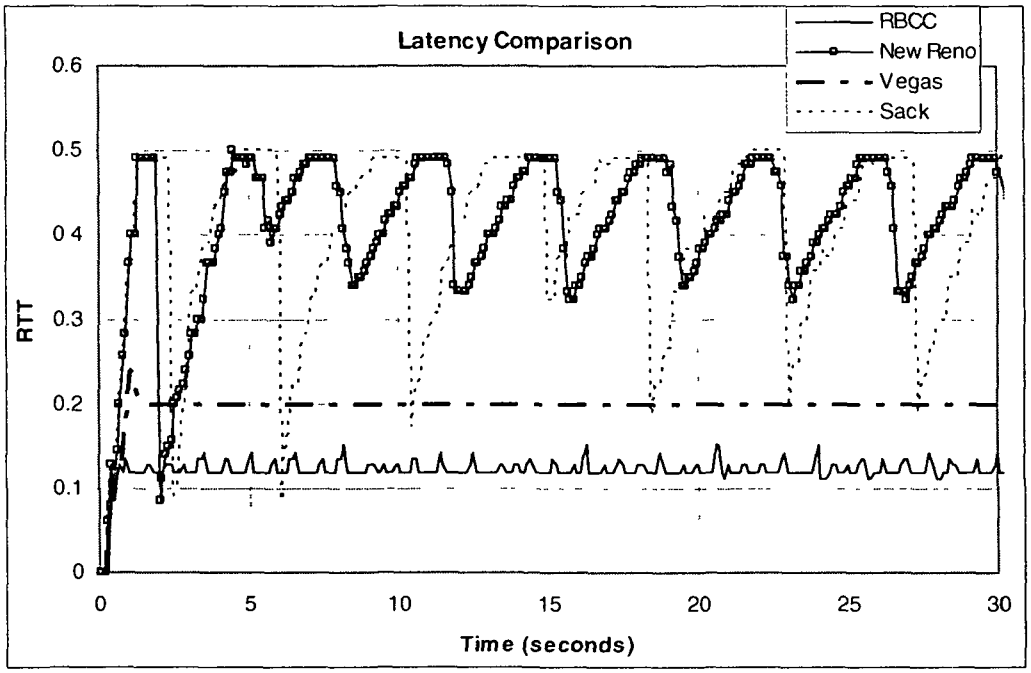

Fig. 4.11 RTT Vs. Time (seconds) in TCP RBCC (F=2.0), NewReno, Vegas and Sack (No. of Connections = 5)

network and the latencies measured in one of the connections against time is plotted. In these comparisons the additional delay due to packet loss and resulting retransmissions

is not taken into account. From the Fig. 4.11, it can be observed that RBCC maintains very low latency which is significantly lower than the other three schemes. The latency for Vegas is closer to that of the new scheme. RBCC also has a much lower jitter than NewReno and SACK. However, the jitter for Vegas is lower than that of RBCC.

4.5.7.5 Queue-Occupancy

Queue-occupancy indicates the memory buffer requirement in the routers. Fig. 4.12 and Fig. 4.13 show the plots of queue occupancy for the four schemes over time in the bottle-neck queue. The plots in Fig. 4.12 have been obtained with 5 simultaneous TCP connections while those in Fig. 4.13 are for increased traffic load with 40 simultaneous TCP connections. It can be seen that the queue occupancy is the lowest for RBCC. For NewReno and SACK the queue occupancy fluctuates heavily and grows very high. At low load the queue occupancy for Vegas is lower than NewReno

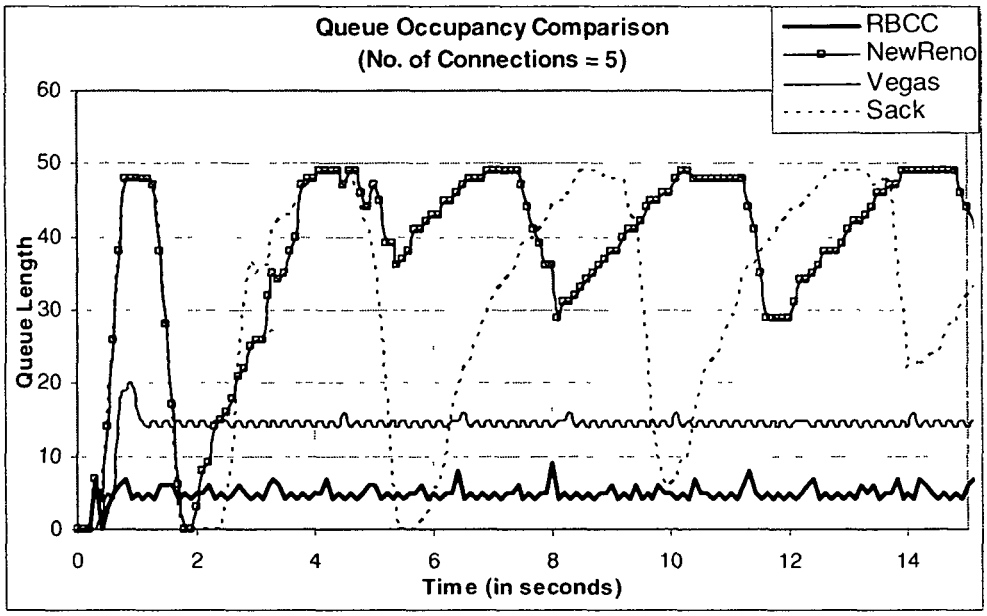

Fig. 4.12 Queue-Occupancy Vs. Time (seconds) in TCP RBCC (F=2.0), NewReno, Vegas and Sack (Number of Connections = 5)

and SACK and its fluctuations are very low. However, at high load the performance of Vegas becomes as poor as the other two schemes. RBCC continues to show much lower queue occupancy at high load while the fluctuations in the occupancy get reduced significantly

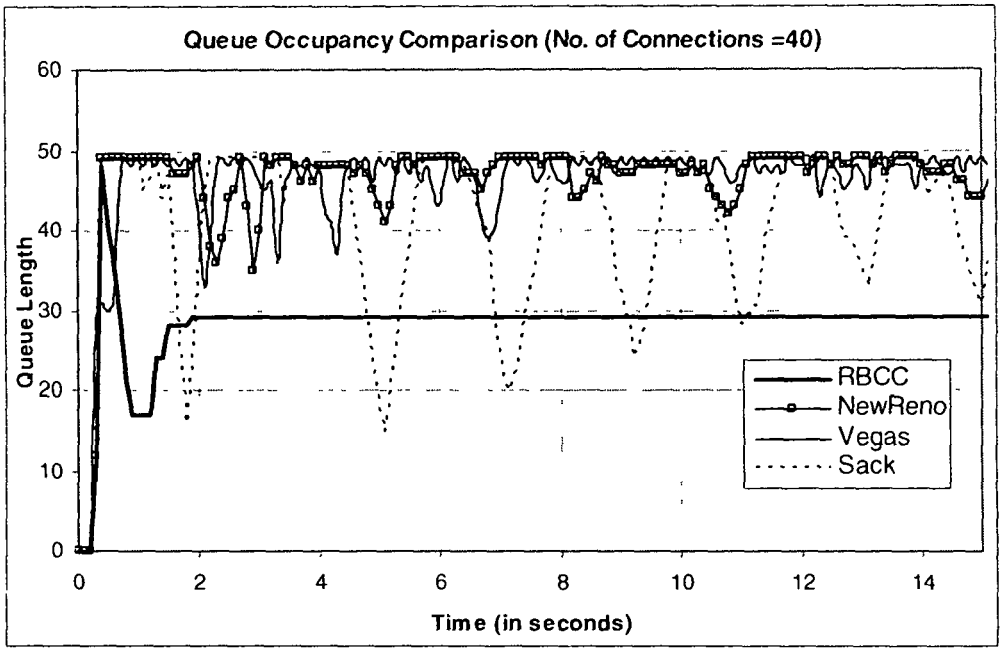

Fig. 4.13 Queue-Occupancy Vs. Time (seconds) in TCP RBCC (F=2.0), NewReno, Vegas and Sack (Number of Connections = 40)

4.5.7.6 Summary of Comparative performance of the Congestion Control Schemes

A Gist of the comparative performance of the four congestion control schemes for TCP, namely, RBCC, NewReno, Vegas, and SACK, are presented in Table 4.1.

Table 4.1
A Gist of Comparative Performance of various TCP congestion control schemes in wired network environment

Scheme	*Packet-Drop (retransmission at source in % of packets Injected)*	*Average Latency (Seconds)*	*Jitter (Standard Deviation)*	*Capacity Utilization (in %)*	*Fairness*
RBCC	*0% (5) (F=2.0)* *0% (25) (F=2.0)*	*0.1241(5) (F=2.0)* *0.1549(5) (F=2.5)*	*0.00908 (5) (F=2.0)* *0.00805 (5) (F=2.5)*	*99% (5) (F=2.0)* *99% (25) (F=2.0)*	*0.95 (5) (F=2.0)* *0.99 (25) (F=2.0)*
Newreno	*1.3% (5)* *12.2% (25)*	*0.4239 (5)*	*0.06158 (5)*	*95% (5)* *92% (25)*	*0.81 (5)* *0.96 (25)*
Vegas	*0% (5)* *8.5% (25)*	*0.1997(5)*	*0.00600(5)*	*99% (5)* *98% (25)*	*0.99 (5)* *0.94 (25)*
SACK	*1.4% (5)* *9.0% (25)*	*0.4003(5)*	*0.09926(5)*	*94% (5)* *94% (25)*	*0.68 (5)* *0.96 (25)*

Note;- 1. The numbers 5 and 25 in the bracket indicate the number of active connections.

4.5.8 Performance of RBCC with Threshold RTT Computed Dynamically in Comparision with Statically Fixed Threshold RTT

So far while comparing RBCC with the existing schemes, its threshold *RTT* was fixed using the static method. In this section we present a study of the comparative performance of RBCC by setting $RTT_{threshold}$ statically and using the dynamic algorithm. The $RTT_{threshold}$ value set in the static method is with $F=2$. For the dynamic algorithm the values of different parameters in the heuristics taken as-

$$\alpha_1 = 12,\ \alpha_2 = 5,\ \beta_1 = 5,\ \beta_2 = 3,\ \gamma_1 = 2,\ \gamma_2 = 2,$$
$$\gamma_{31} = 5,\ \gamma_{32} = -5,\ \gamma_{41} = 15,\ \gamma_{42} = 15,\ \gamma_{51} = 15,\ \gamma_{52} = 15.$$

4.5.8.1 Packet Loss

Fig. 4.14 shows the plots for packet-loss for increasing traffic load. It can be seen that performances of RBCC (Static) and RBCC (Dynamic) are similar up to a reasonably high traffic load. Only at very high traffic load situation the packet losses in RBCC (Dynamic) grow higher than RBCC (Static).

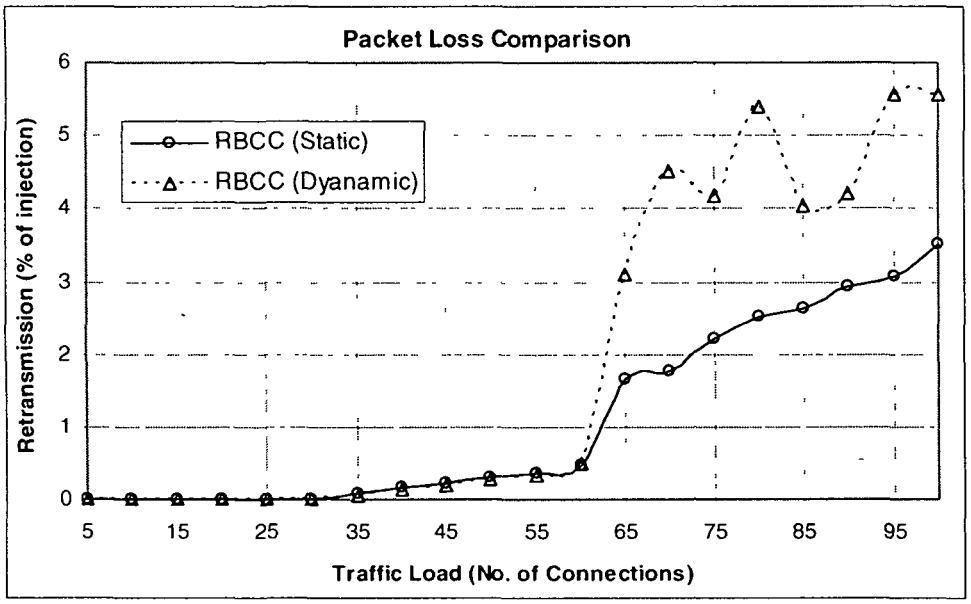

Fig. 4.14 Packet Loss Vs. Traffic Load in TCP RBCC with Static (F=2.0) and Dynamic Threshold RTT

4.5.8.2 Throughput

Fig. 4.15 shows the plots of throughput in RBCC(Static) and RBCC(Dynamic) for increasing traffic load. It can be seen that there is no visible difference in the

performance of the two schemes in terms of throughput. In both the cases the capacity utilization is near 100%.

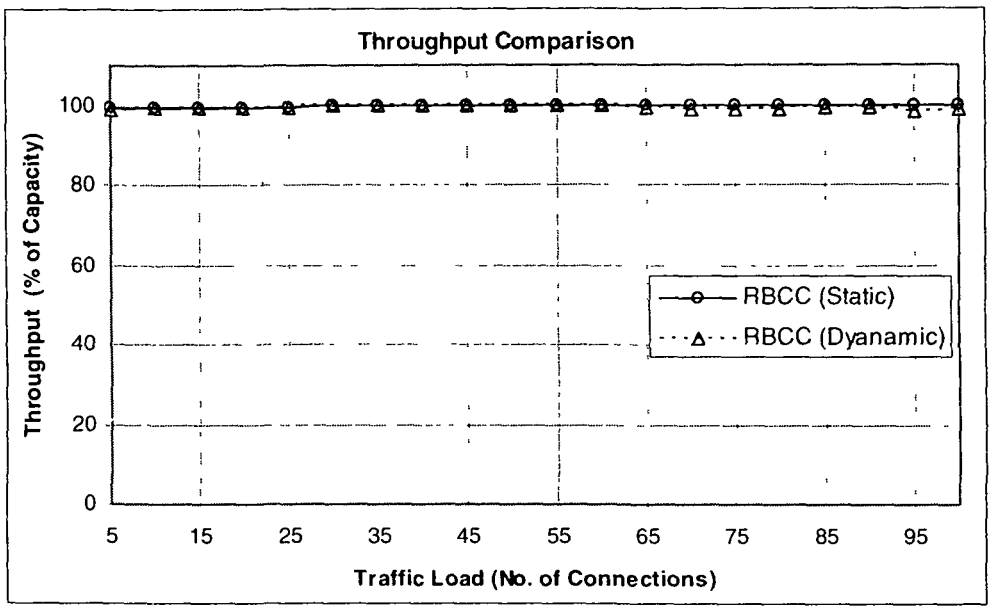

Fig. 4.15 Throughput Vs. Traffic Load in TCP RBCC with Static (F=2.0) and Dynamic Threshold RTT

4.5.8.3 Fairness

Fig.4.16 shows the plots for fairness of RBCC(Static) and RBCC(Dynamic) against varying traffic load. It can be observed that in terms of fairness both the methods perform equally well most of the time. Only at low load the dynamic algorithm shows somewhat lower fairness than the static one.

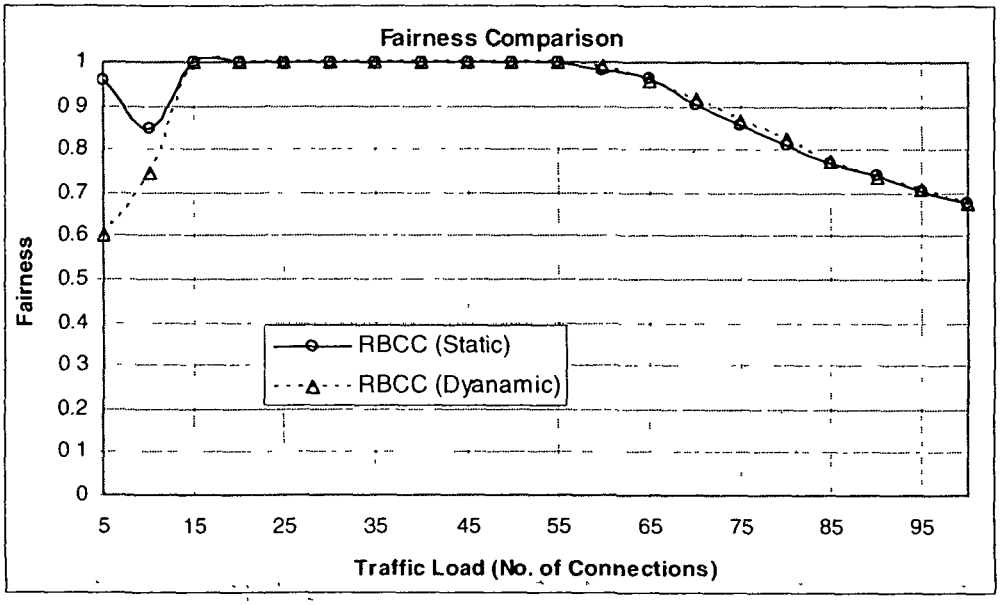

Fig. 4.16 Fairness Vs. Traffic Load in TCP RBCC with Static (F=2.0) and Dynamic Threshold RTT

4.5.8.4 Packet Latency

Fig. 4.17 shows the plots for packet latency for RBCC(Static) and RBCC(Dynamic) against time for a fixed number of connections. It can be observed that the dynamic algorithm is able to maintain a marginally lower latency than the static one.

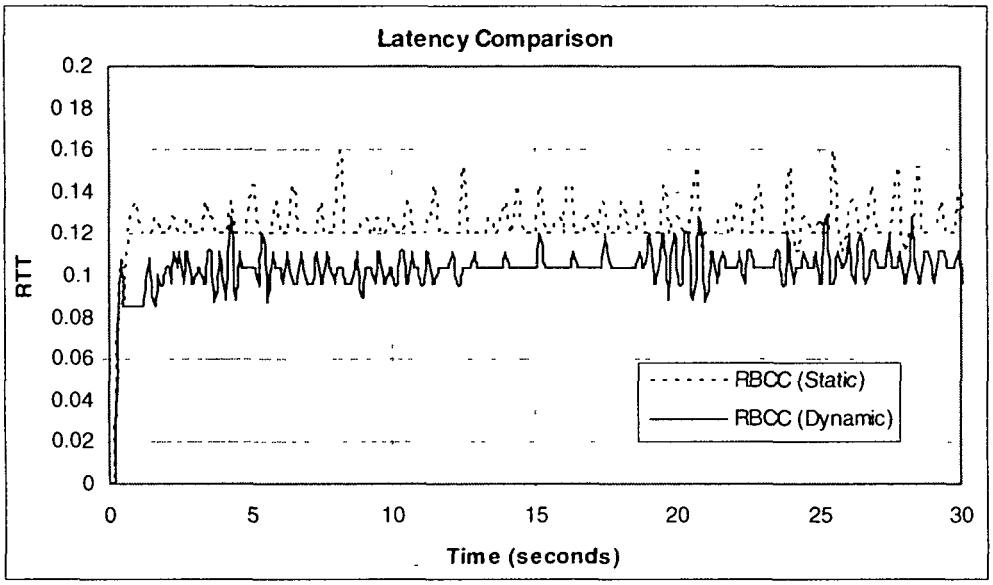

Fig. 4.17 RTT Vs. Time in TCP RBCC with Static (F=2.0) and Dynamic Threshold RTT (No. Connections =5 and RTTs are taken for the first flow in each case)

4.5.8.5 Queue-Occupancy

The queue-occupancy plot for RBCC(Static) and RBCC(Dynamic) against time are shown in Fig. 4.18. It can be observed that the dynamic algorithm is able to maintain a marginally lower queue-occupancy than the static one.

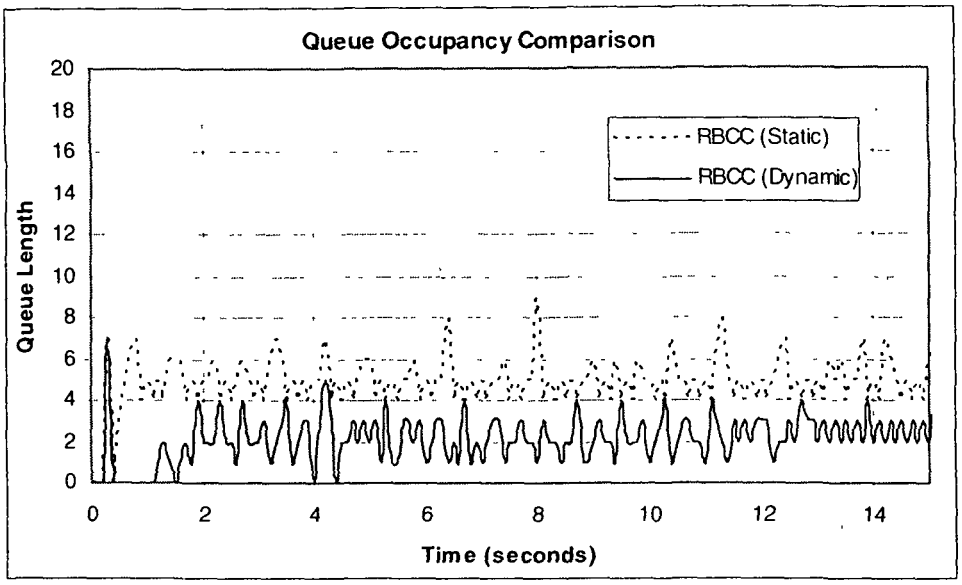

Fig. 4.18 Queue-Occupancy Vs. Time in TCP RBCC with Static (F=2.0) and Dynamic Threshold RTT (No. Connections =5)

4.6 Summary

In this chapter, the TCP RBCC scheme as applied in Wired Network has been presented. Methods for setting the threshold RTT statically as well as through dynamic computation have also been presented. Results of experiments carried out in simulated environments demonstrate the ability RBCC to address TCP's congestion control problems by keeping low queue-length and low packet-drops. The performance of the scheme has been compared with TCP NewReno, Vegas and SACK. It has been shown that RBCC performs better than these three existing schemes in terms of packet-loss, throughput, latency, queue-occupancy and fairness. The RBCC does a better job because it eliminates the main flaw in traditional TCP (e.g. Newreno, Sack etc.) viz. primary mechanism for detection of congestion is packet drop and reduce the congestion window size always on packet drop. This mechanism can be termed as reactive approach. RBCC, rather, adopts the proactive approach by taking corrective measures when congestion starts building up in the bottle-neck link. It uses RTT as a control parameter for congestion as congestion is reflected in RTT by means of the queuing delay. Though Vegas reduces *cwnd* (if the difference between actual and expected throughput is greater than beta parameter) without depending upon packet drop, but its parameters don't seem to reflect congestion accurately. Because the expected throughput invariably remains the same whatever could be the dynamics in the

subnet. Also the calculation of actual throughput is done per RTT, whereas RBCC does it more granularly i.e. per ACK.

The performance of RBCC with dynamic computation of Threshold RTT has been shown to be at per with that done statically; though the dynamic algorithm some time shows lower performance than that of static. With the static fixing of the Threshold RTT the experiments were carried out with threshold values fixed over a range to show that desired performance can be achieved by appropriately fixing the threshold value. For one or more of the threshold values the scheme produces the best results. Dynamic fixing of the threshold is required so that the scheme adopts the appropriate Threshold RTT value on its own to produce the desired performance. In an ideal case the threshold value chosen will be such as to produce the best possible performance. However, dynamic scheme uses heuristics and it only tries to come as close to the ideal solution as possible.

On the other hand, dynamic algorithm could be better answer for the scenario where there is possibility of changing route dynamically in a lifetime of a connection. Though the experiment could not be conducted for such scenarios in this study, with the dynamic fixing of the Threshold RTT, due to the mechanism for monitoring the conditions of the network and for adopting the appropriate threshold value, it is expected that the schemes will perform satisfactorily even when the topology changes. This however is not the case for static fixing of the Threshold RTT as it is required to fix the threshold value as appropriate for the topology.

Chapter 5

Adaptation of the RTT Based Scheme to Networks with Wireless Links

In this chapter, an adaptation of the RTT based congestion control scheme for TCP, discussed in Chapter 3, to networks with wireless links is presented. This adaptation particularly addresses the issue of non-congestion related packet-losses that are prevalent in wireless environment. Simulation based experimental results comparing the performance of this adaptation with NewReno, Vegas, SACK and Snoop are also presented.

5.1 RTT in Networks with Wireless Links

In a wired network where all the links are wired, most of the packet-losses are due to congestion, RTT varies only due to queuing delay. In a network with wireless links, however, apart from congestion related losses, there are frequent occurrences of losses due to signal fading, packet-corruption, bit-errors etc. This causes network performance to degrade and there is drop in bandwidth utilization efficiency. To counter this, some mechanisms are adapted in the MAC Layer of Wireless Networks. For example, IEEE 802.11 uses acknowledgement (ACK) mechanism at the MAC layer. If the sender does not receive an ACK frame within some period of time, it retransmits the data frame. The number of retransmission is, however, limited. There is also provision for exchange of two control packets, RTS and CTS between the sender and receiver, before exchanging a data packet to address the hidden terminal and exposed terminal problems. The Bluetooth, on the other hand, uses strong FEC (Forward Error Correction) scheme that increases overhead on MAC layer for processing the data frames and introduces significant amount of delay. It also uses ARQ (Automatic Repeat Request) technique that retransmits a data frame if its ACK is not received. The MAC Layer, in Wireless Networks, entailing these

additional functionalities lead to a high delay component in RTT. The delays introduced by FEC and RTS/CTS are generally constant. However, the delay due to the ACK mechanism in the link layer depends much on the link state. Whenever there is a link-layer retransmission there is a jump in the delay. Therefore the RTT on Wireless Networks (involving the MAC Layer such as IEEE 802.11 or Bluetooth) does not only vary due to queuing delay but also due to the retransmission delay. Thus the *RTT* in this case can be expressed as:

$$RTT_{wl} = t_{tr} + t_{pd} + t_{pr} + t_{fe} + t_{qd} + t_{rd} \qquad \text{-------- (5.1)}$$

Where t_{fe} is the constant delay due to processing of the routine tasks such as FECs and RTS/CTS and t_{rd} is the variable delay due to retransmission at link-layer. The remaining components are being same as those in equation-(3.1).

Now, there are two variable delays components in RTT viz. t_{qd} and t_{rd}. In this circumstance, RTT will not be able to reflect the congestion scenario properly as t_{rd} is not due to congestion. If RBCC is to be used in wireless environment, the MAC Layer should be without the DATA/ACK and retransmission mechanism. Therefore a simple MAC Layer without the DATA/ACK and retransmission mechanism is considered for employing RBCC. The RTT with a simple MAC layer is then-

$$RTT_{wl} = t_{tr} + t_{pd} + t_{pr} + t_{fe} + t_{qd} \qquad \text{-------- (5.2)}$$

In this t_{qd} is the only variable component and therefore can be relied on as a measure of congestion.

5.2 Proposed Adaptation of RBCC to Address Wireless Issues

The adaptation of RBCC for Wireless Network which will be referred to as RBCC-WL consists of the following steps:

i) Initialize the size of *cwnd* to one segment at the time of connection setup

ii) Compute fine-grained RTT using TCP Timestamp option for each packet on reception of it's ACK packet

iii) Maintain a Threshold RTT value which may be calculated statically or dynamically.

iv) On reception of each ACK increment the size of *cwnd* by-

$$cwndIncr = \frac{(RTT_{threshold} - RTT)}{max\,(RTT, RTT_{threshold})} \; X\; Segment\; Size \qquad \text{---------- (5.3)}$$

v) Do not reduce the size of *cwnd* on packet-drop.

vi) Follow TCP NewReno for detection of packet-loss, retransmission time-out estimation, fast retransmit and fast recovery.

There are two changes in the adaptation. The first, in the denominator of the expression for computation *cwndIncr,* in equation (5.3), the larger of the two values of *RTT* and $RTT_{threshold}$ is taken instead of the smaller one. The absolute value of *cwndIncr* becomes smaller here. Though, it makes *cwnd* increment or decrement little more conservative, it helps in having lesser delay-jitters and more stable queue-length. It is also related to the second change, i.e. *cwnd* is not reduced on detection of packet loss. Here being aggressive in *cwnd* increment/decrement is likely to backfire.

One of the main issues with the TCP has been treating all types of packet loss as losses due to congestion. TCP reduces the size of the *cwnd* to by either half or one depending upon how the loss was detected. In RBCC-WL the packet losses due to congestion are rare as extreme congestion situation is not allowed to occur. Most packet losses are, therefore, wireless-related. As congestion is handled independently, the reduction in *cwnd* is therefore not required on detection of packet-loss. Other than these, the new scheme follows TCP Newreno for *cwnd* initialization to one segment-size at the beginning of a connection, in detection of multiple packet-losses in a single-window and for fast retransmit.

5.3 Experimental Results

Experiments have been carried out in a Simulated Wireless Environment using NS2 to observe the performance of the new scheme (RBCC-WL) in comparison to TCP Newreno, Vegas, Sack and Snoop.

5.3.1 Topology and Environment for Experiments

The topology used for the experiments is shown in Fig.5.1, where every source-sink pair (Src_i - $Sink_i$, $i = 0$ *to* 4) is connected through a common 1 Mbps link (R-BS) with 10 ms propagation delay. Each of the feeding links also has a bandwidth of 1 Mbps and 10 ms propagation delay. The host nodes connected with wireless links are allowed to be mobile in a topology covering an area of 600m by 600m. These mobile nodes are connected to the Base Station (BS) through a 2 Mbps wireless-channel having omni type antenna. The ad-hoc routing protocol used for mobile nodes is DSDV [PB94]. The wireless interface queue type is Drop Tail and queue buffer limit is configured as 50. The radio-propagation model used for the wireless link is *shadowing* [Rap03], [FV00] as it is the closest model for real environment. The *shadowing deviation,* the standard deviation of a Gaussian random variable with zero mean used in the *shadowing model* that reflects the variation of the received power over varying distances, is kept constant at value 4.0. The *reference distance,* the close-in distance in free space that is used with actual distance to compute the average path loss, is taken as 1.0 m [Rap03]. The *Path Loss Exponent (PLE)* [Rap03] determines the extent of the wireless related packet losses and its value is kept within the range [2-2.7] that reflects the obstructions in a factory scenario.

All the experiments were carried out with the simple MAC Layer for the wireless link. In this, the MAC layer does not use Forward Error Correction (FEC) and link level retransmissions (ARQ). Therefore *RTT* can be assumed to have only the queuing delay as the variable component.

The wireless nodes are mobile and allowed to move around within a given range. For the data generation at the sink an *FTP* transfer of a very large file is done in each connection. The queuing discipline used is *Drop Tail* with queue buffer limit of 50. Every simulation is run for 50 seconds duration. The traffic patterns are changed by starting and terminating the TCP connections from the Fixed Hosts to the Wireless Mobile Hosts.

Experiments have been conducted to observe the various aspects such as- ability of the scheme to control packet-loss, performance of RBCC-WL in comparison to the existing schemes in terms of throughput, fairness, packet latency and delay jitter. In these experiments the $RTT_{threshold}$ value is fixed statically as a product of a factor F *(>0)* and RTT_{min} *(i.e.* $RTT_{threshold} = F \times RTT_{min}$, *where* RTT_{min} is the minimum estimated *RTT* that pre-computed based on link delay, hops, propagation time and processing time*)*. The variation in $RTT_{threshold}$ is achieved by varying F.

TCP window size is set to 20, three number of duplicate ACKs are used for Fast Retransmit, TCP clock granularity is set to 0.01 seconds, the slow start threshold is set to 20 and immediate ACK is performed as per RFC [APS99]. These parameters are common across all the variants of TCP (including the new scheme). For TCP Newreno (and also the RBCC), slow-but-steady variant is used as per the RFC [FH99] with retransmit timer reset after each partial new ACK. Also, *cwnd* is set to *ssthresh* upon leaving fast recovery and upon partial ACK (i.e. no window deflation option) [FH99]. For TCP Vegas the values of

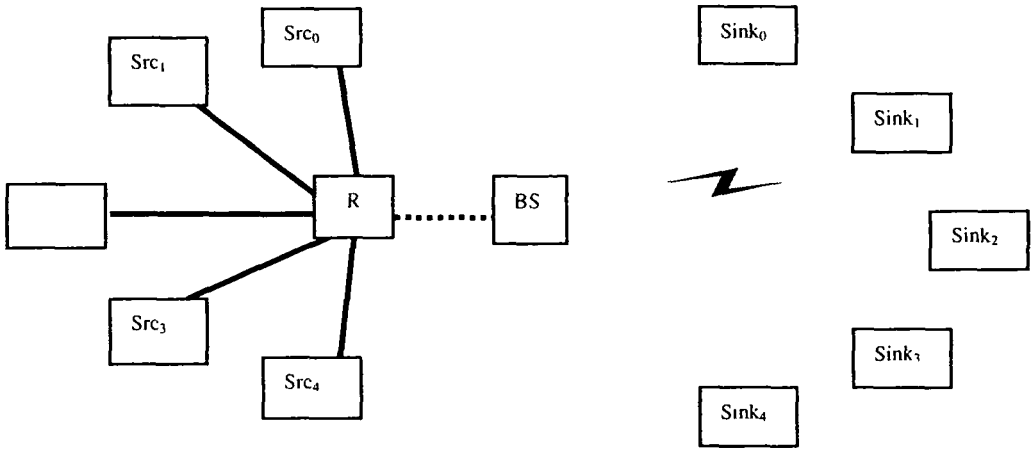

Fig. 5.1 Wireless Topology

Alpha, Beta and Gamma parameters are set to 1, 3 and 1 respectively, which are default. For TCP SACK, maximum SACK blocks is set to 3. For Snoop, maximum packet buffer size is set to 100 and maximum retransmission is configured to be 10.

5.3.2 Ability of the Scheme to Control Congestion Related Packet-losses

The first experiment was to verify the ability of the scheme to control congestion related packet loss. It was carried out for *PLE* values of 2.1 and 2.3. The results are presented in

Fig. 5.2. Ten FTP connections were set up over TCP RBCC-WL and the data injection rate was varied by varying the $RTT_{threshold}$ value. It can be observed in Fig. 5.2 that the packet losses remain low up to a certain $RTT_{threshold}$ value (represented by F). After $RTT_{threshold}$ exceeds this value the packet-losses start growing fast as the threshold becomes too high to be effective. In the initial portion where the threshold value is very low the packet injection rate is too low. For PLE=2.3 the packet losses are higher as the path obstruction become high in this case.

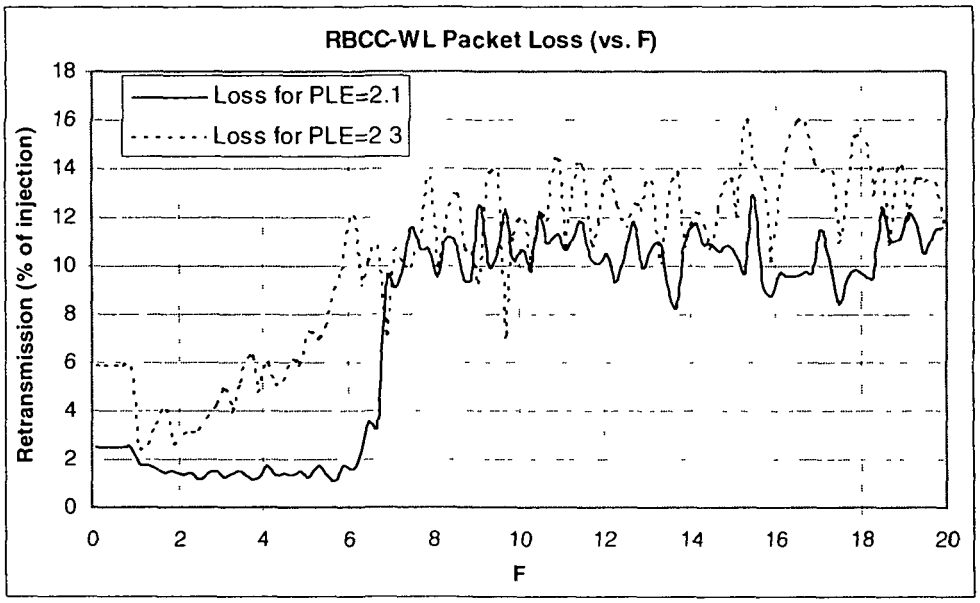

Fig.5.2 Packet Loss vs. F in RBCC-WL (for No. of Connections = 10)

5.3.3 Comparison of Throughput, Packet-loss and Fairness with TCP Newreno, Vegas, SACK and Snoop

Simulations were carried out to compare the throughput and packet-loss behavior of the new scheme (RBCC-WL) with TCP Newreno, Vegas, SACK and Snoop against varying values of the path loss exponent (PLE). For the experiments with Snoop, TCP Newreno was used at the source nodes (i.e. Fixed Hosts) and the Snoop Agent was attached at the Base Station (BS) node. The wireless related losses increase with increased values of PLE. The comparison of Fairness was done against varying number of TCP connections.

The experiments were conducted on simple MAC Layer. Fig. 5.3 shows plots of the throughput (against PLE) for TCP RBCC-WL, Newreno, Vegas, Sack and Snoop. Though for initial values of PLE, RBCC-WL and Vegas show similar throughout, RBCC-WL shows better performance than all the others for PLE up to 2.4. For PLE higher than 2.4 the performance of RBCC-WL falls below Snoop, but it continues to perform better than the rest of the schemes.

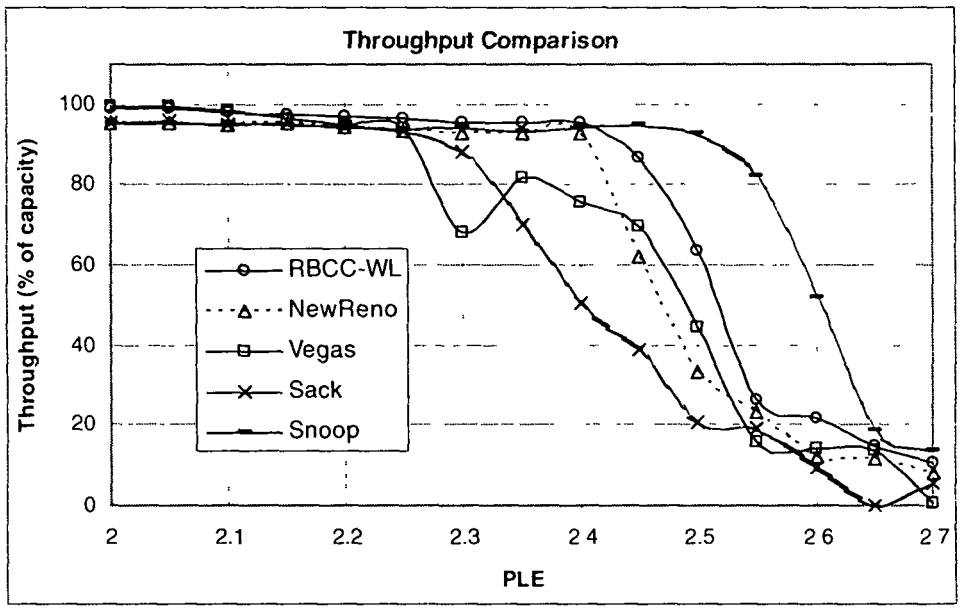

Fig.5.3 Throughput in RBCC-WL, Newreno, Vegas, SACK and SNOOP (No. of Connections = 10, F=1.7)

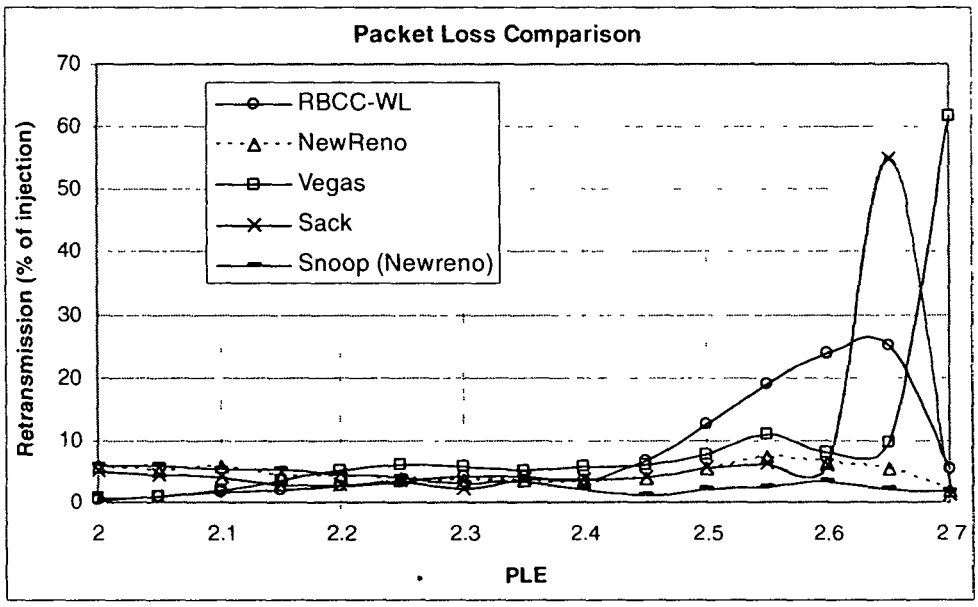

Fig.5.4 Packet Loss (Retransmissions) in RBCC-WL, Newreno, Vegas, SACK and Snoop (No. of Connections = 10, F=1.7)

Fig. 5.4 shows that the packet losses detected at source (retransmissions at source as percentage of packets injected). In RBCC-WL it is lesser than in Newreno, Vegas, SACK as well as Snoop except at high values of PLE. For high values of PLE, RBCC-WL's packet losses are higher as most of these losses are due to wireless link degradation and RBCC-WL continues to inject packets without reducing *cwnd* size. For Snoop most of the packet losses are not detected at source as the wireless related losses are tackled by packet retransmission from the BS node.

The fairness graphs plotted in Fig. 5.5 shows that RBCC-WL is fairer than TCP Sack. It is also fairer than the other three except for small number of connections. In fact RBCC-WL grows fairer with increasing number of connections before its fairness starts to fall gradually for very large number of connections.

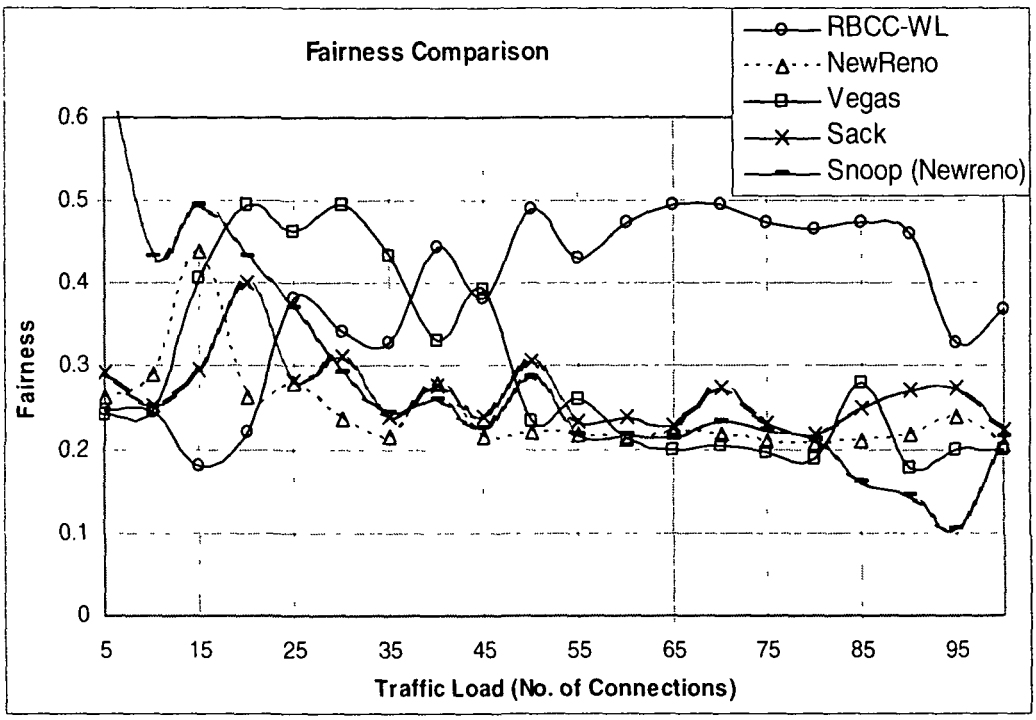

Fig.5.5 Fairness in RBCC-WL, Newreno, Vegas, SACK and Snoop (No. of Connections = 10, F=1.7, PLE=2.2)

5.3.4 Packet Latency and Delay Jitter

Packet Latency and delay jitter for RBCC-WL, TCP Newreno, Vegas, Sack and Snoop in terms of RTT are plotted in Fig. 5.6. It can be seen that here Snoop performs very poorly. Both latency and jitter for TCP Snoop are significantly higher than the other four schemes. One reason for this is that the RTT includes the retransmission delay from the BS node. In latency the performance of RBCC-WL and Vegas are more or less similar. Compared to New Reno and Sack latency and jitter are significantly lower for RBCC-WL. Compared to Vegas, RBCC-WL performs better in terms of delay jitter.

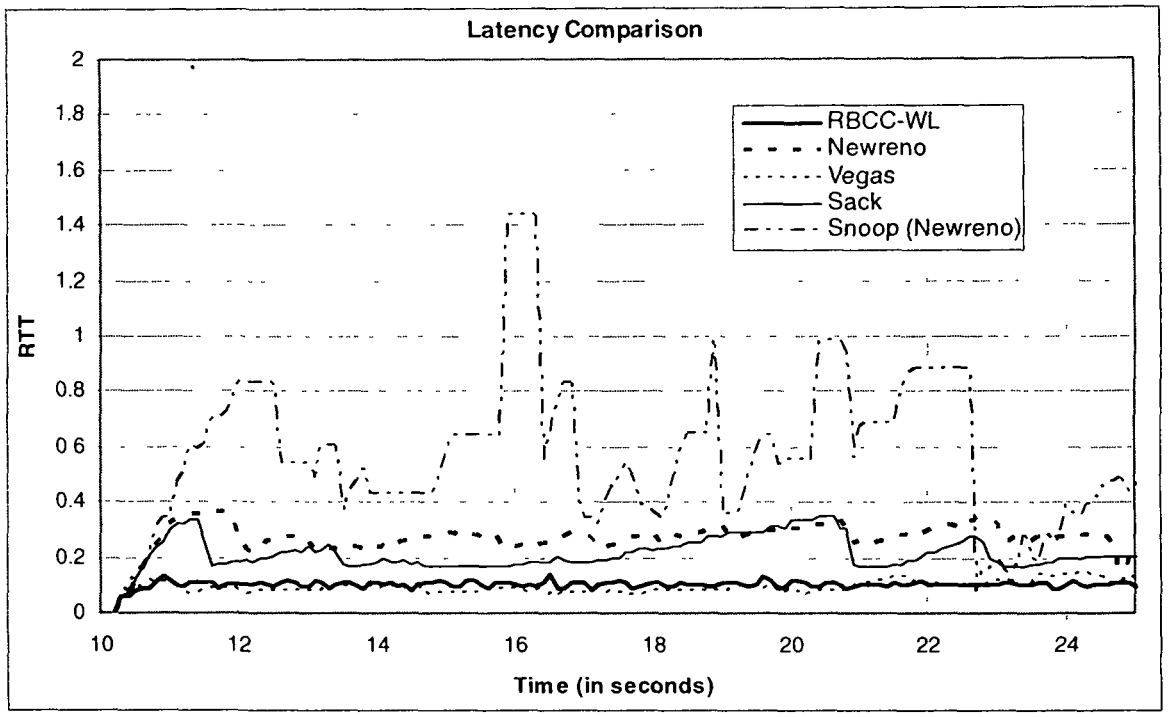

Fig.5.6 Latency in RBCC-WL, Newreno, Vegas, SACK and Snoop (No. of Connections = 5, F=1.7, PLE=2.2)

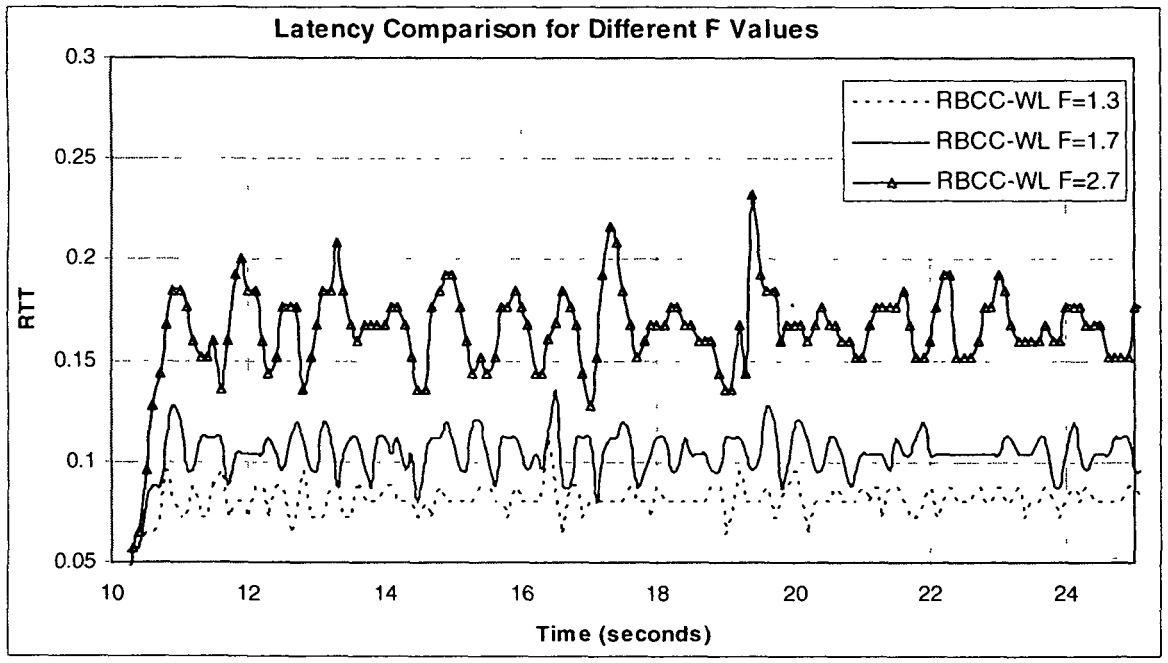

Fig.5.7 Latency in RBCC-WL for different values of F (1.3, 1.7 and 2.7) (No. of Connections = 5, PLE=2.2)

It is also possible to control the packet latency and delay jitter in RBCC-WL by choosing appropriate value for $RTT_{threshold}$ (in this case by adjusting the F value) as can be seen in Fig. 5.7. For low F values it maintains lower latencies and jitter. Thus these QoS parameters can be set in RBCC-WL as desired.

5.3.5 Congestion Window Comparison

To understand the behavior of the schemes the variations in the *congestion window* size in time for the schemes are plotted in Fig. 5.8. It can be observed that for Newreno, Sack and Snoop the *congestion window size* grows very high and comes down suddenly whenever packet drops are detected. RBCC-WL and Vegas maintain a steady level and achieve a steady throughput. Fig. 5.9 shows plots for *congestion window* in RBCC-WL for different $RTT_{threshold}$ values.

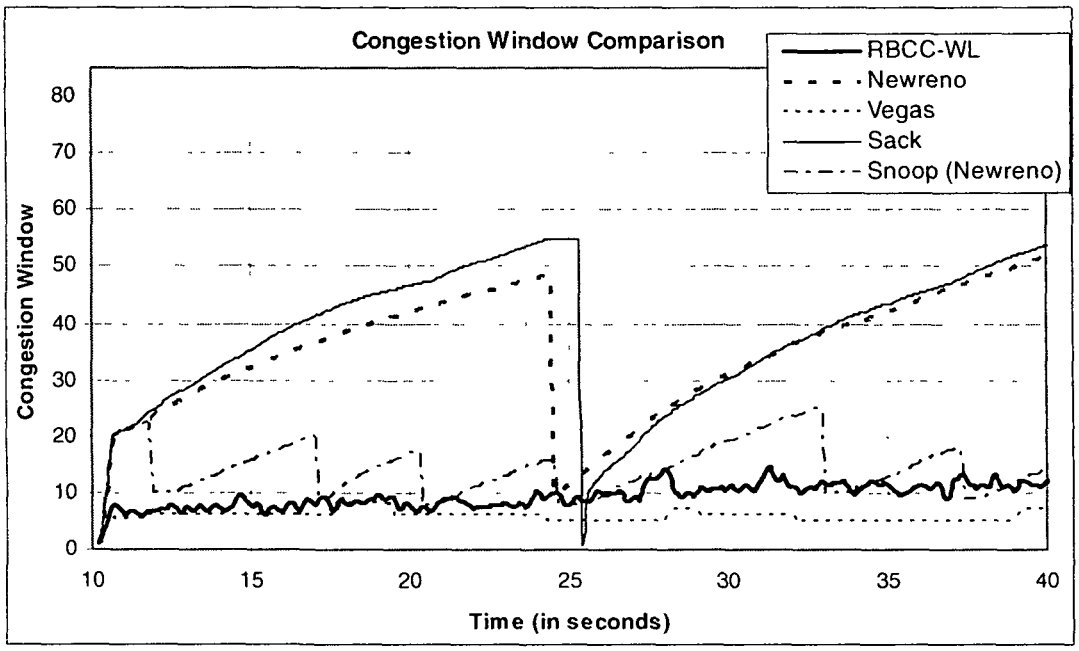

Fig.5.8 Congestion Window in RBCC-WL, Newreno, Vegas and SACK (No. of Connections = 5, F=1.7, PLE=2.2)

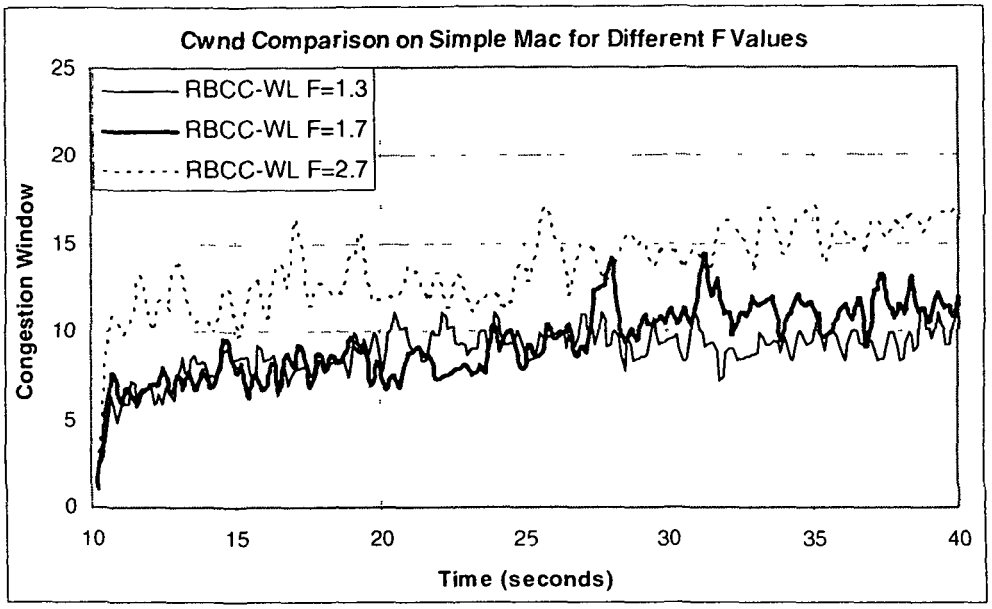

Fig.5.9 Congestion Window in RBCC-WL for different values of F (1.3, 1.7 and 2.7) (No. of Connections = 5, PLE=2.2)

5.4 Summary

In this chapter, an adaptation of TCP RBCC for Wireless Network has been presented. The RTT for wireless scenarios has been analyzed and two changes in TCP RBCC have been proposed for the modified version referred to as TCP RBCC-WL. The changes proposed are- non-reduction of *cwnd* size on packet-drop and making the congestion window increment less aggressive by replacing the minimum function by a maximum function in the *cwndIncr* expression. The first modification helps in avoiding unnecessary reduction in *cwnd* as the congestion is controlled independently. The second change makes the absolute value of *cwndIncr* little more conservative and helps in reducing the delay-jitters. The results of the simulation experiments presented demonstrate RBCC-WL's ability to address the problems of TCP related to wireless networks by making the congestion detection independent of the packet loss. The performance results of the scheme have been compared with TCP NewReno, Vegas, SACK and Snoop and have been shown that RBCC-WL performs better than the first three schemes in terms of throughput, latency and fairness, which are summarized in Table 5.1. TCP Snoop shows better throughput than RBCC-WL when wireless related losses are high. However, it performs very poorly in terms of fairness, latency and delay jitter.

Table 5.1

A Gist of Comparative Performance of Various TCP Congestion Control Schemes in Wireless Environment under Different Error Conditions (Path Loss Exponent (PLE))

Scheme	*Packet-Drop (retransmission at source) (% of Injected Packets)*	*Average Latency (Seconds)*	*Jitter (Std. Dev.)*	*Capacity Utilization (in %)*	*Fairness*
RBCC-WL	*0.43% (2.0) (F=1.7)* *2.4% (2.2) (F=1.7)*	*0.1050 (2.2) (F=1.7)* *0.1176 (2.2) (F=1.9)*	*0.00874 (2.2) (F=1.7)* *0.01077 (2.2) (F=1.9)*	*99% (2.0) (F=1.7)* *97% (2.2) (F=1.7)*	*0.24 (5)* *0.38 (25)* *0.48 (50)*
Newreno	*5.9% (2.0)* *4.1% (2.2)*	*0.2672 (2.2)*	*0.04999(2.2)*	*95% (2.0)* *94% (2.2)*	*0.26 (5)* *0.27 (25)* *0.22 (50)*
Vegas	*0.49% (2.0)* *4.9% (2.2)*	*0.1047 (2.2)*	*0.02707(2.2)*	*99% (2.0)* *95% (2.2)*	*0.24 (5)* *0.46 (25)* *0.23 (50)*
Sack	*5.2% (2.0)* *2.9% (2.2)*	*0.2109 (2.2)*	*0.04238 (2.2)*	*95% (2.0)* *94% (2.2)*	*0.29 (5)* *0.28 (25)* *0.30 (50)*
Snoop (with Newreno)	*6.1% (2.0)* *4.6% (2.2)*	*0.5749 (2.2)*	*0.22330 (2.2)*	*95% (2.0)* *94% (2.2)*	*0.67 (5)* *0.37 (25)* *0.28 (50)*

Note: 1) The numbers 2.0 and 2.2 within brackets indicate the PLE. Higher PLE means higher bit-errors.

2) The numbers 5, 25 and 50 in bracket are the number of connections in the fairness column

Chapter 6

New Congestion Control Schemes for SCTP

In this chapter, we present two new congestion control scheme for SCTP. The first one, called RTT Based Congestion Avoidance (RBCA), is an adaptation of the *RTT based congestion scheme (RBCC) for TCP presented in Chapter 3. The second* scheme called Switch Path on Congestion (SPC) exploits the multihoming feature of SCTP. A comparative study of performance of both these schemes with the standard version of SCTP has been carried out through simulation experiments. The results of this study are also presented in this chapter.

6.1 Proposed Schemes for SCTP

6.1.1 Adaptation of RBCC to SCTP

The scheme basically involves monitoring of *RTT* for the packets transmitted on each path and increasing or decreasing the rate of injection of packets into the network by appropriately adjusting the *Congestion Window* size depending on the *RTT* value falling below a $RTT_{threshold}$ value or crossing over it. To decide on the increment / decrement required in the *Congestion Window* size the equation (6.1) below is used. The values of *RTT* and $RTT_{threshold}$ are maintained for each path. The *Congestion Window* size is updated as follows:

i. Compute fine-grained RTT on receipt of each SACK using the Timestamp Control Chunk.
ii. As in standard SCTP *cwnd* is changed only if the current *cwnd* is fully utilized and the incoming SACK advances cumulative ACK point.
iii. On receipt of SACK on a path, compute *CwndIncr,* the *Congestion Window* increment as per the expression below:

$$CwndIncr = \frac{(RTT_{threshold} - RTT)}{max(RTT, RTT_{threshold})} X\ DSize \quad \text{---------- (6.1)}$$

Where

DSize = min(NewlyAckedBytes, MaxDataSize)

NewlyAckedBytes is the number of newly ACKed Bytes in a SACK

MaxDataSize is the Maximum Packet Size

iv. Compute *Cwnd as-*

$$Cwnd = Cwnd + CwndIncr. \quad \text{----------(6.2)}$$

If computed *Cwnd* < *MaxDataSize* then *Cwnd* is set to *MaxDataSize.*

v. Do not reduce *Cwnd* on packet drops.

vi. Do not use Delayed ACKs [1].

The changes in the expression for *cwndIncr* with reference to equation 3.4 are-

a. *Segment Size* in equation 3.4 is replaced by *DSize* in equation 6.2. *DSize* is nothing but the size by which the standard SCTP changes *cwnd.* Unlike in standard SCTP, here *cwnd* is changed by fraction of *DSize* proportionate to the difference between *RTT* and $RTT_{threshold}$, not by *DSize.*

b. In the denominator of the expression the larger one of *RTT* and $RTT_{threshold}$ is chosen instead of the smaller. This is similar to the equation 5.3 in the wireless adaptation. Like in the wireless case, this becomes a little conservative in *cwnd* change but compensates the fact that *cwnd* is not reduced on packet-loss. It also helps SCTP in improving on fairness and delay jitter.

Further, in the modified scheme for SCTP *cwnd* is not reduced on packet drop and delayed ACK is not used. The reason for not reducing *cwnd* is that with appropriate fixing of $RTT_{threshold}$ value extreme congestion situations are avoided. Any packet drop will very likely be due to reasons other than congestion. The reason for the delayed ACK not being used is that it leads to *RTT* values that do not reflect the queuing delays correctly.

In the following this new scheme for SCTP will be referred to as *RTT Based Congestion Avoidance (RBCA).*

6.1.2 Switch Path on Congestion (SPC) technique for utilizing the alternate path in SCTP

This scheme observes the back-to-back *RTT*s to determine the traffic status of the *primary path.* If the *RTT* increases consecutively and crosses the $RTT_{threshold}$ then the primary path is assumed to be approaching congestion. In such situation the traffic is switched to the *alternate path* making it the *primary path.* The steps involved in this scheme referred to as *Switch Path on Congestion (SPC)* are as follows:

i. On receipt of an SACK update *RTT,* the previous *RTT (PRTT)* and the *RTT* prior to *PRTT (PPRTT)*
ii. If on the *primary path* the *RTT* grows consecutively for two SACKs and lies above $RTT_{threshold}$, i.e. if $PPRTT > RTT_{threshold}$ and $RTT > PRTT$, $PRTT > PPRTT$ then
    a. Make *alternate path* as the *primary path.*
    b. Initialize *RTT, PRTT and PPRTT* to 0 on the *new primary path.*
    c. Set the *Cwnd* in the *new primary path* as-

$$Cwnd = max(OldCwnd \times P, 2 \times MaxDataSize) \qquad \text{----------(6.3)}$$

Where,

$$P = \frac{(3 + floor\ (HowOld / 10))}{max(7, HowOld)} \qquad \text{----------(6.4)}$$

In equation-(6.4) *OldCwnd* is the congestion window of the *new primary path* before the switching. *HowOld* is a measure for the duration for which the *primary path* (prior to switching) was continuously in use. The measure is in count of the number of SACK chunks received during the period. This variable is initialized on each path switch.

The expression *P* above is made to lay in a range *0.1<P<0.44* so that it shrinks as the value of *HowOld* increases. The numbers 3, 10 and 7 used in the expression have been chosen experimentally to achieve the best performance. The value of *HowOld* specifies the freshness of the *new primary path* and indicates whether resumption of the transmission is being made after short period or long period of time. Low value means high freshness and high value means its being idle for some time. Thus, on resumption of transmissions on the *new primary path*, *cwnd* will be reduced by a factor inversely proportional to the length of time this path was idle.

Since the proposed schemes need continuous monitoring of the current *RTT* value, it uses Timestamp option which adds a 12-byte Timestamp chunk into every packet with DATA or SACK chunk(s) [CAS03]. This allows *RTT* measurements to be made per packet on both original transmission and retransmissions.

6.1.3 Resulting Schemes

The use of RBCA and SPC lead to the following three new schemes for congestion control in SCTP:

1. SCTP with RBCA - *SCTP (RBCA),*
2. SCTP with SPC - *SCTP (SPC),*
3. SCTP with both RBCA & SPC – *SCTP (RBCA & SPC).*

In the first case SCTP will use RBCA for congestion control, i.e. adjustments of the congestion window will be done as per RBCA. In the second case SPC will be used with standard SCTP. That is, the congestion window adjustments will be made as for normal SCTP, but the primary path will be changed based on the congestion status reported by the SPC Technique. In the third case the congestion window adjustments will be made as per RBCA and the primary path will be changed based on the congestion status reported by SPC Technique.

6.2 Experimental Results

A series of experiments have been carried out to verify the advantage of using RBCA and SPC for SCTP For the experiments the following cases of SCTP are considered-

1 SCTP with the standard congestion control mechanism - *SCTP(Std)*,

2 SCTP with RBCA - *SCTP (RBCA)*,

3 SCTP with SPC - *SCTP (SPC)*,

4 SCTP with both RBCA & SPC – *SCTP (RBCA & SPC)*

6.2.1 Topology and Environment for Experiments

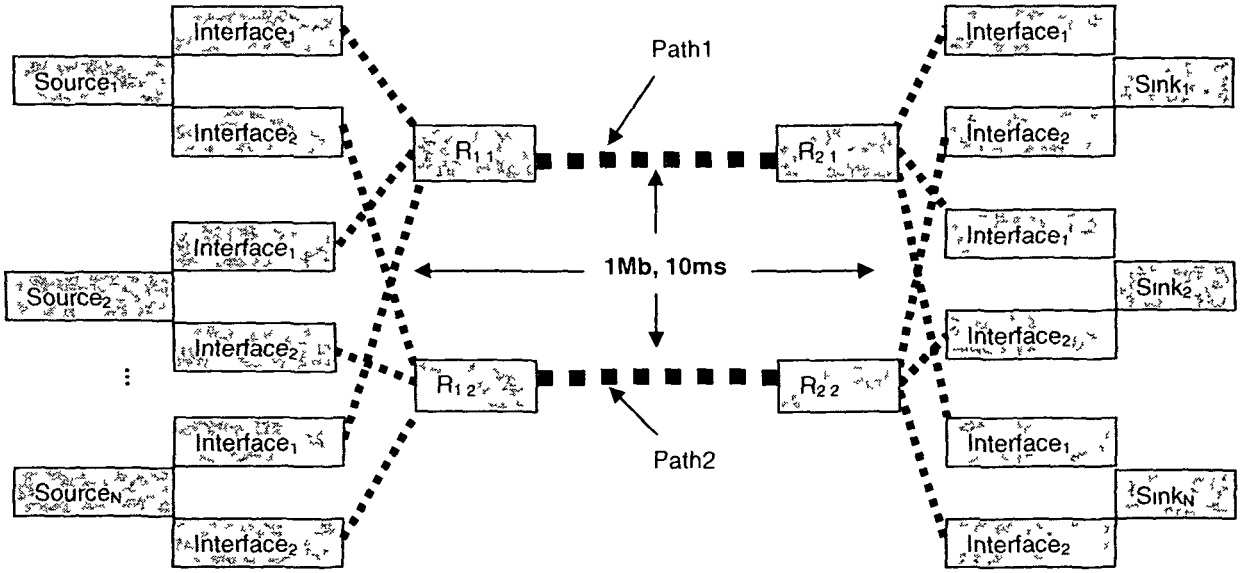

Fig 6 1 Topology for SCTP in NS2

The experiments have been carried out using the Network Simulator package *NS* [NS2] on Linux platform The topology shown in Fig 6 1 was used for the experiments Here every source-sink node pair (Source$_i$ – Sink$_i$, i=*1 to N*) is connected through two parallel paths each having a 1 Mbps bottleneck link These are $R_{1\,1}$-$R_{2\,1}$ and $R_{1\,2}$-$R_{2\,2}$, each with 10 ms propagation delay Each of the feeding links has a bandwidth of 1 Mbps and 10 ms propagation delay Each of the end nodes (i e either source or sink node) has two interfaces (Interface$_1$ and Interface$_2$) one each for the two paths Thus, each of the end nodes can connect through two parallel paths using two separate feeding

links through two separate interfaces. In the experiments the data packets are generated by running an *ftp* application on a very large file at every source node where each of the simulation is run for 200 seconds and the primary path is switched at 20 second. The queuing discipline used is Drop-Tail with buffer limit of 50. A few experiments are also conducted with different bit error rates, to see how the different versions of SCTP perform in a scenario where there are non-congestion related packet losses. For this, unit of error is taken as 'packet' and the random variable for generating errors is uniformly distributed from 0 to 1.

Experiments have been conducted to observe the various aspects such as- ability of the scheme to control packet-loss, to maintain good throughput and fairness and to be able to keep latency and jitter low. In the experiments for RBCA the $RTT_{threshold}$ value is fixed statically as-

$$RTT_{threshold} = F \times RTT_{min} \quad \text{---------- (6.5)}$$

where RTT_{min} *is* computed a priori based on propagation time, transmission time and processing time along the path to the destination node as per equation (3.2).

The variation in $RTT_{threshold}$ was achieved by varying the constant factor F.

While comparing packet-drop, throughput and fairness, the experiments were carried out as follows. Multiple *ftp* connections were set up and the traffic load was increased by increasing the number of source-sink pairs. The $RTT_{threshold}$ was kept fixed with F=2.0 for the three new schemes of SCTP. While taking account of bandwidth utilization, the sum of the bandwidths of both of the paths has been considered. The throughput and the packet-drops were taken as aggregate of the flows. Here no non-congestion related losses are introduced.

The different parameters of SCTP [Ste00] are configured as following:

i. Association.Max.Retrans = 10 attempts
ii. Path.Max.Retrans = 5 attempts (per destination)
iii. Max.Init.Retransmits = 8 attempts
iv. Heartbeat Interval = 30 seconds
v. MTU = 1500 (MTU of Ethernet, most common)
vi. Initial Receiver Window = 1310720
vii. Initial *Ssthresh* = 65536

viii. Initial *cwnd* = 2 * MTU
ix. Initial RTO = 3 secs
x. Minimum RTO = 1 sec
xi. Maximum RTO = 60 secs
xii. Number of missing reports to trigger fast retransmit = 4
xiii. Data Chunk Size = 1456
xiv. Use Delayed Sacks = Yes, but no for SCTP RBCA
xv. Sack Delay = 200 ms
xvi. By default retransmission to alternate destination = Yes

6.2.2 Packet Losses due to Congestion

The first experiment was to verify the ability of the scheme to control packet losses. The comparison has been made between the four versions of the SCTP viz. SCTP (Std.), SCTP (RBCA), SCTP (SPC) and SCTP (RBCA & SPC). As shown in the plots in Fig. 6.2, SCTP (Std.) displays highest number of packet drops. As the traffic load increases the losses grow very rapidly for SCTP(Std.). On the other hand, SCTP (RBCA) shows very less number of packet drops. SCTP (RBCA & SPC) also shows similar results. In case of SCTP (SPC) the packet-drops significantly lower than SCTP(Std.).

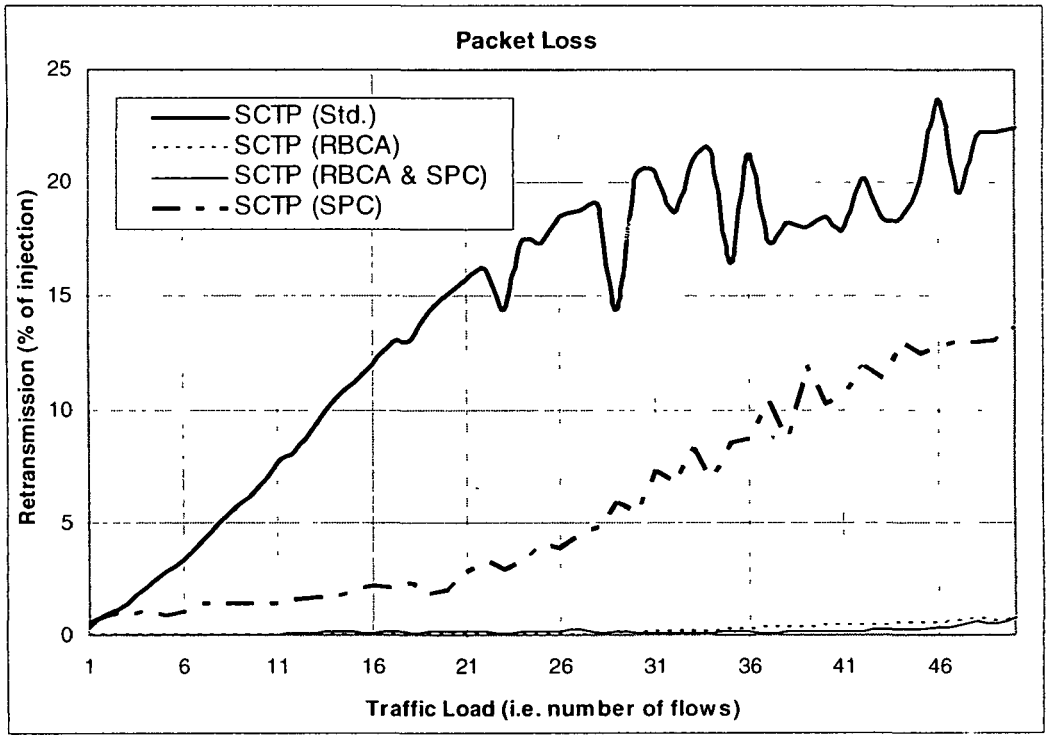

Fig. 6.2 Packet Loss comparison between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC)

6.2.3 Throughput

A comparison of the throughput achieved by the four versions is shown in Fig. 6.3. The throughput is lowest in case of SCTP (RBCA). It achieves throughput equal to the capacity of one of the paths. SCTP (Std.) shows some what higher throughput than SCTP (RBCA) and it grows with traffic load. This happens due to use of the alternate path for the retransmissions. With traffic load packet losses grow thus increasing the retransmissions through the alternate path. Therefore the throughput grows in SCTP(Std.) with traffic load. On the other hand, SCTP (RBCA) has very less retransmission of packets and uses the alternate path very infrequently, while the primary path is used to the fullest.

When SPC is used SCTP achieves much higher throughput. It nearly doubles, i.e. the capacity of both the paths, primary and alternate get utilized. With SPC whenever congestion approaches there is a switching of the paths. Thus packets are injected alternately into the two paths. Both the paths are used nearly to the fullest of the capacity. Both SCTP (RBCA & SPC) and SCTP (SPC) show similar throughput. Both achieve a high degree of load balancing when traffic load is high.

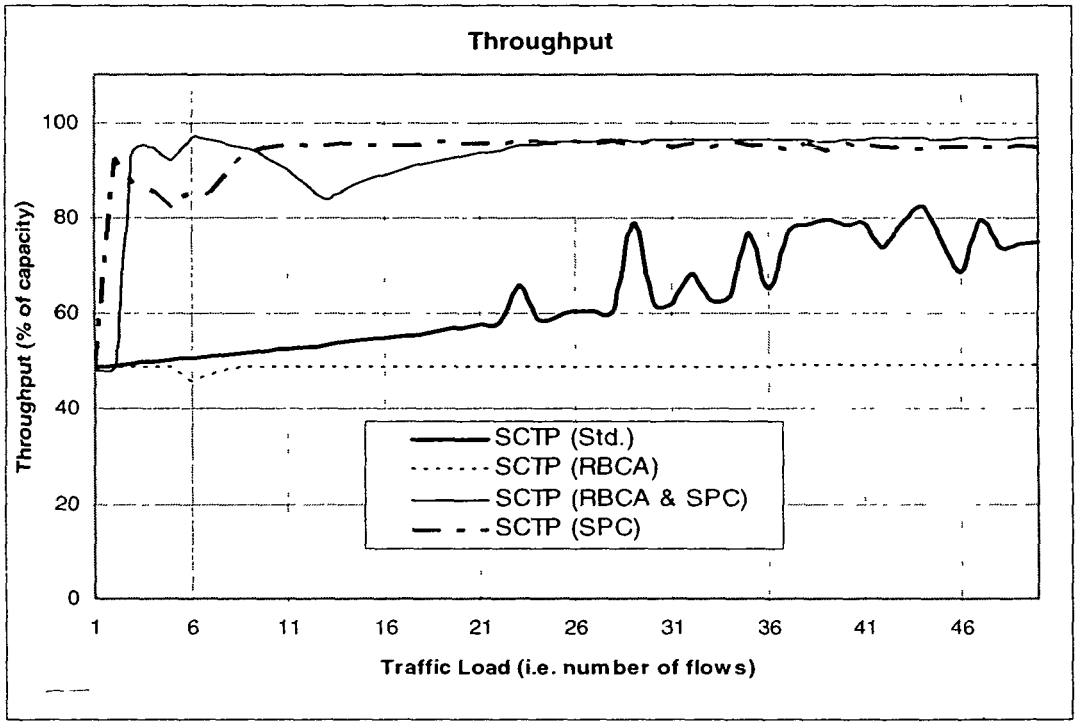

Fig. 6.3 Throughput comparison between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC)

6.2.4 Fairness

As shown in the plots in the Fig. 6.4, as the traffic load increases SCTP (Std.)'s fairness drops significantly. SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC) demonstrate similar high fairness even at high traffic loads.

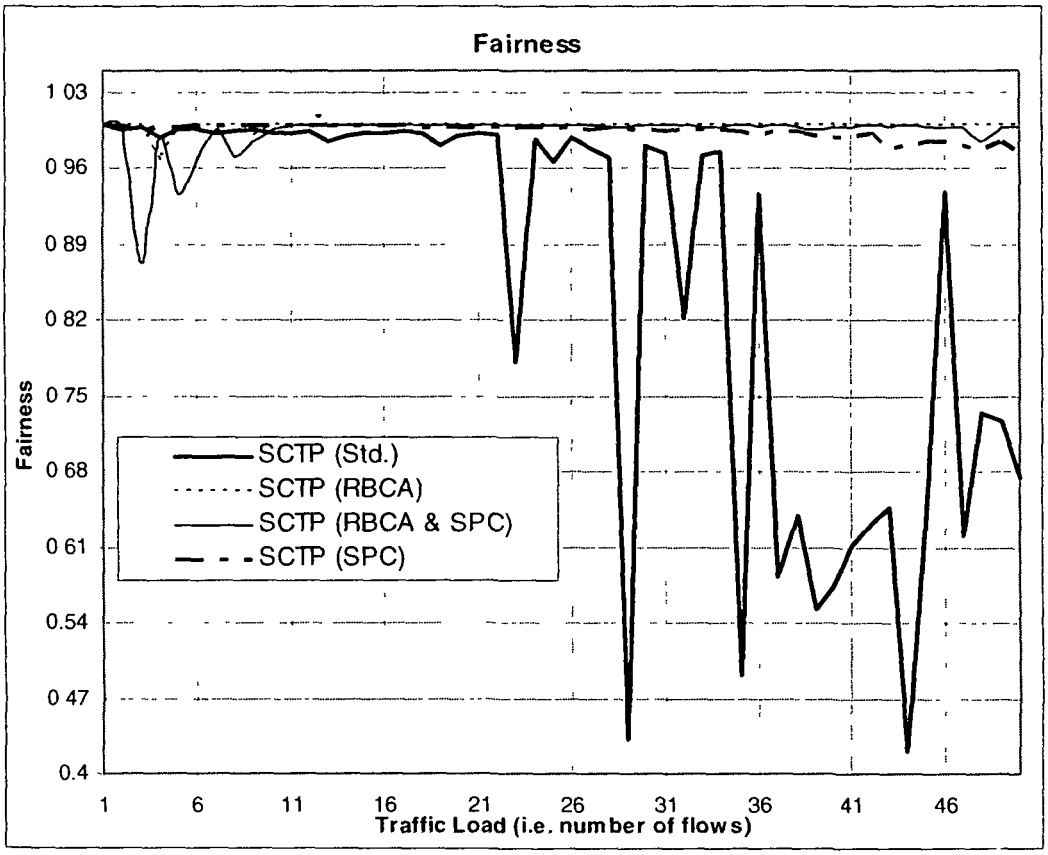

Fig. 6.4 Fairness comparison between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC)

6.2.5 Packet Latency and Jitter

The comparison of the delay and delay jitters suffered by packets is made between all the four versions of SCTP in Fig. 6.5. SCTP (Std.) demonstrates high delay and jitters, whereas SCTP (RBCA) and SCTP (RBCA & SPC) maintain very low delay and jitters. Here SCTP (SPC) hangs in the middle, reiterating that SPC has left its mark on latency too.

Fig. 6.6 and Fig. 6.7 show the delay suffered by packets in SCTP (RBCA) and SCTP (RBCA & SPC) for different values of F. From this it can be observed that the

delay and the delay jitter can be controlled by choosing appropriate value for $RTT_{threshold}$.

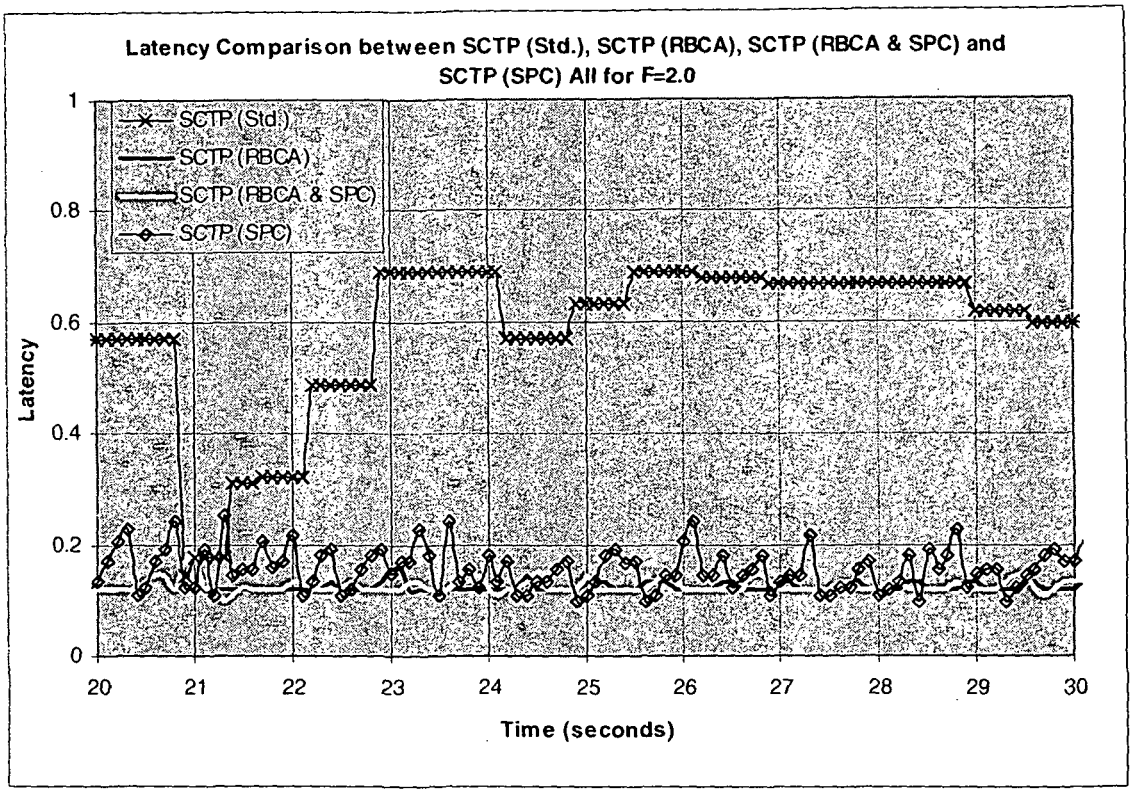

Fig. 6.5 Packet Latency comparison between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC)

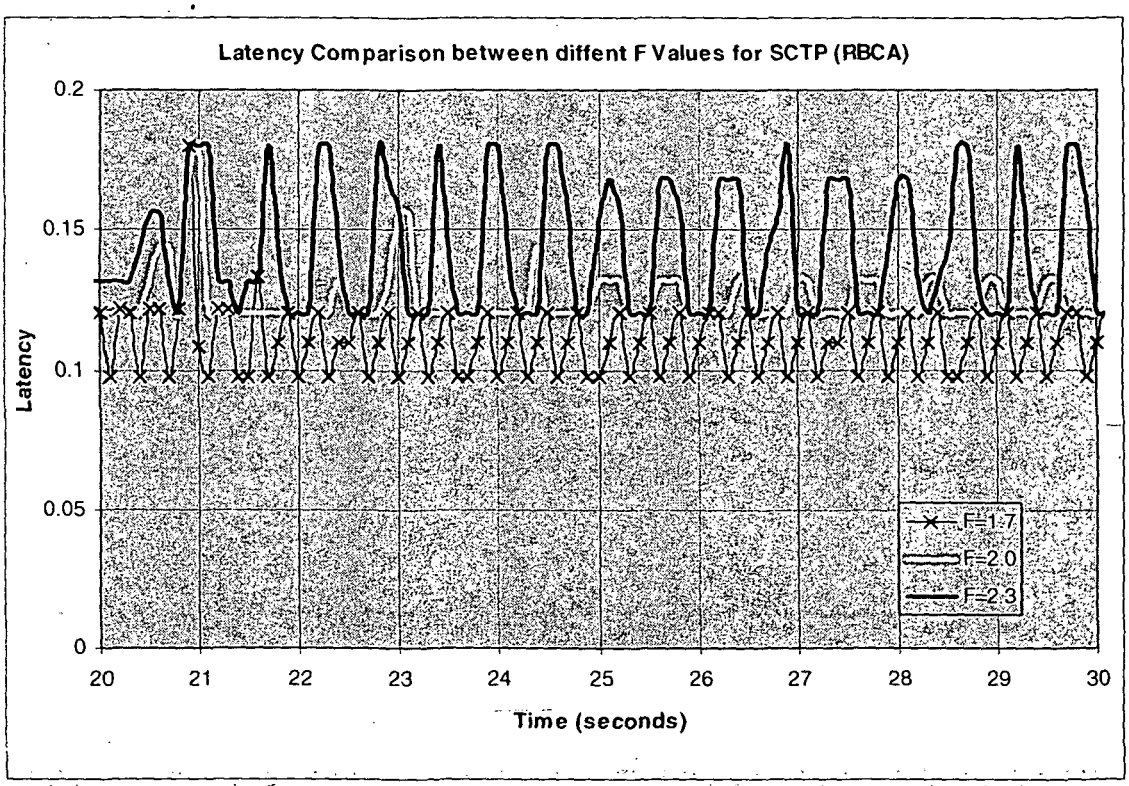

Fig. 6.6 Packet Latency comparison between different Values of F in SCTP (RBCA)

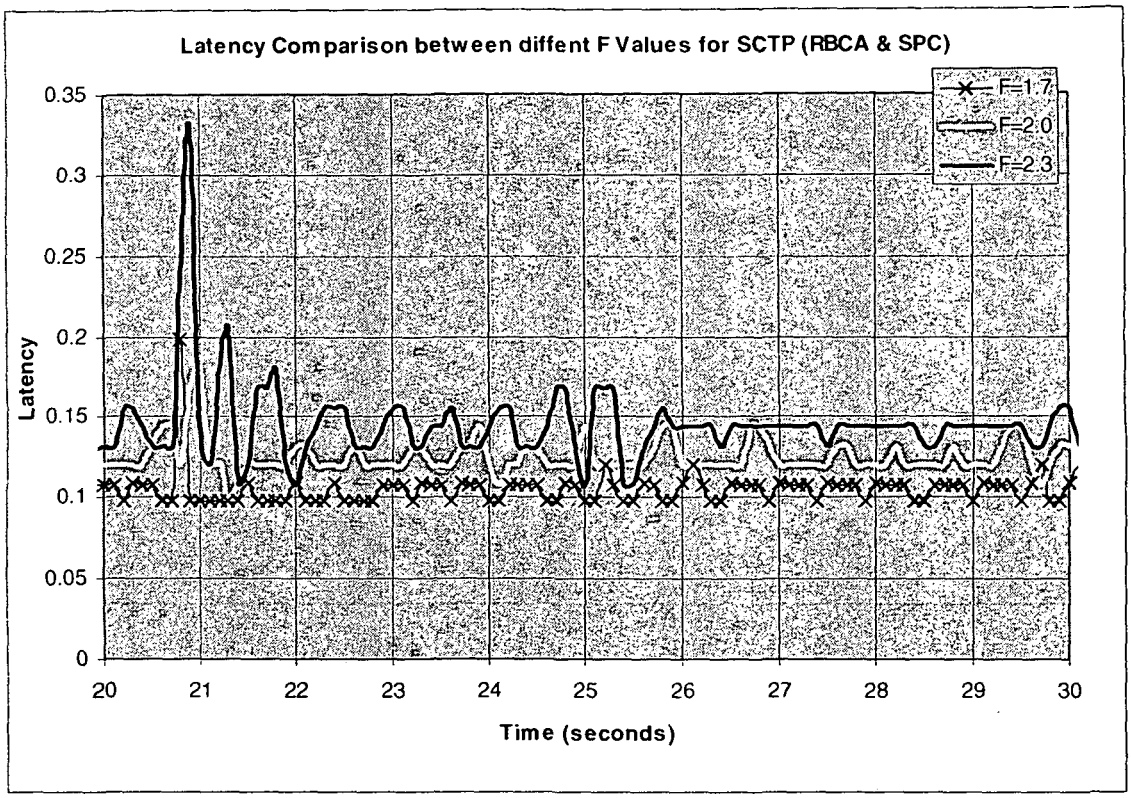

Fig. 6.7 Packet Latency comparison between different Values of F in SCTP (RBCA & SPC)

6.2.6 Queue Occupancy

The comparison of the queue-occupancy at the bottle-neck link of Path 2 is shown in Fig. 6.8. The comparison is made between all the four versions of SCTP. SCTP (Std.) demonstrates high queue-occupancy, whereas the other three schemes maintain very low queue-occupancy. It is lowest for SCTP (RBCA & SPC). The time in the plot is put from 20^{th} second onwards as the primary path was switched to Path 2 after 20 seconds of the simulation run.

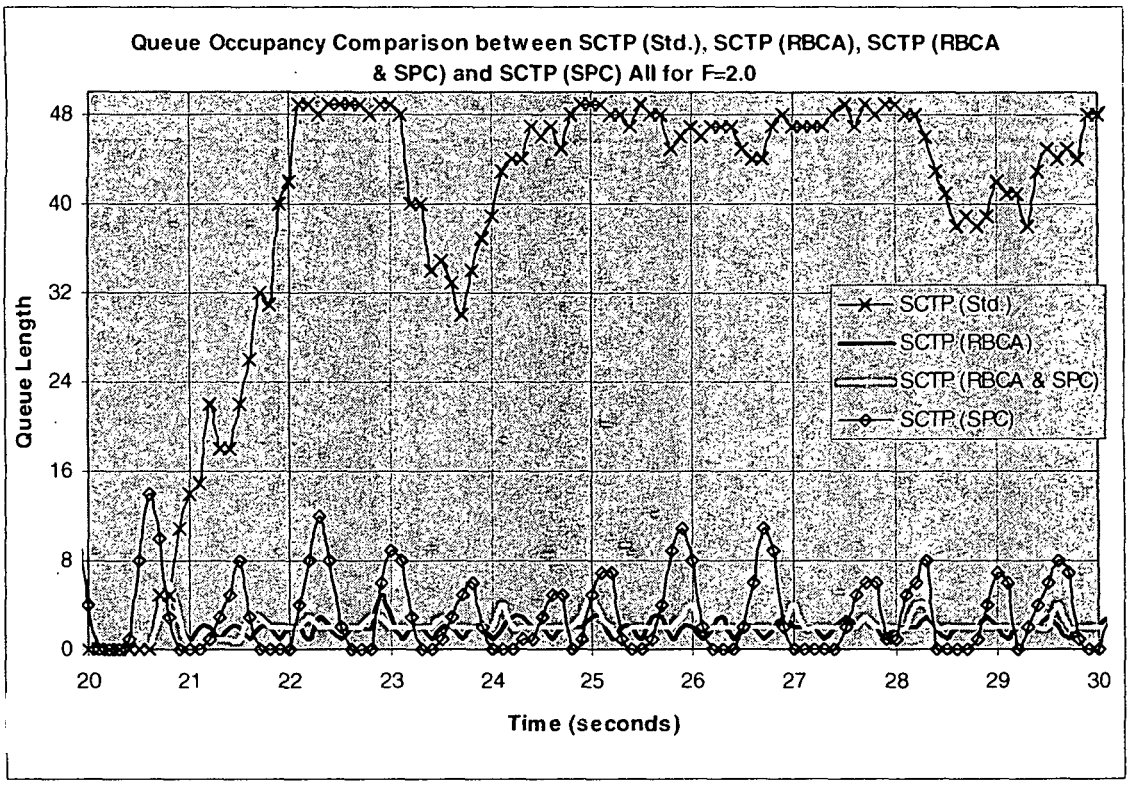

Fig. 6.8 Comparison of Queue Occupancy (in Path2) between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC)

6.3 Experiment With Packet Losses due to Bit Errors

A set of experiments were also carried out to observe the effect of bit errors in the path on the different versions of SCTP. These experiments have been conducted with a single connection between a source-sink pair. Bit errors with different error rates were introduced and the effects on the throughput were observed. For RBCA the F value was set to 2.0. The results of these experiments are shown in Fig. 6.9 to Fig. 6.13. It can be observed that as the error rate increases SCTP (Std.)'s performance degrades rapidly. The degradation in performance for SCTP with RBCA & SPC is very gradual.

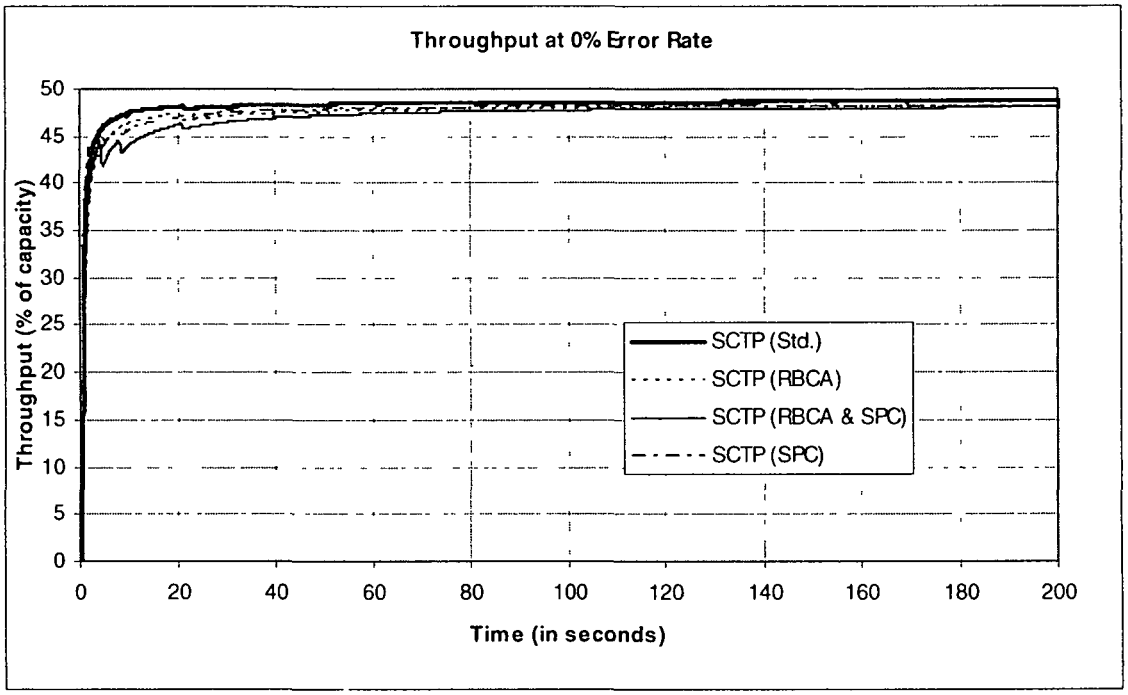

Fig. 6.9 Throughput comparison between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC), F=2.0 with 0% error

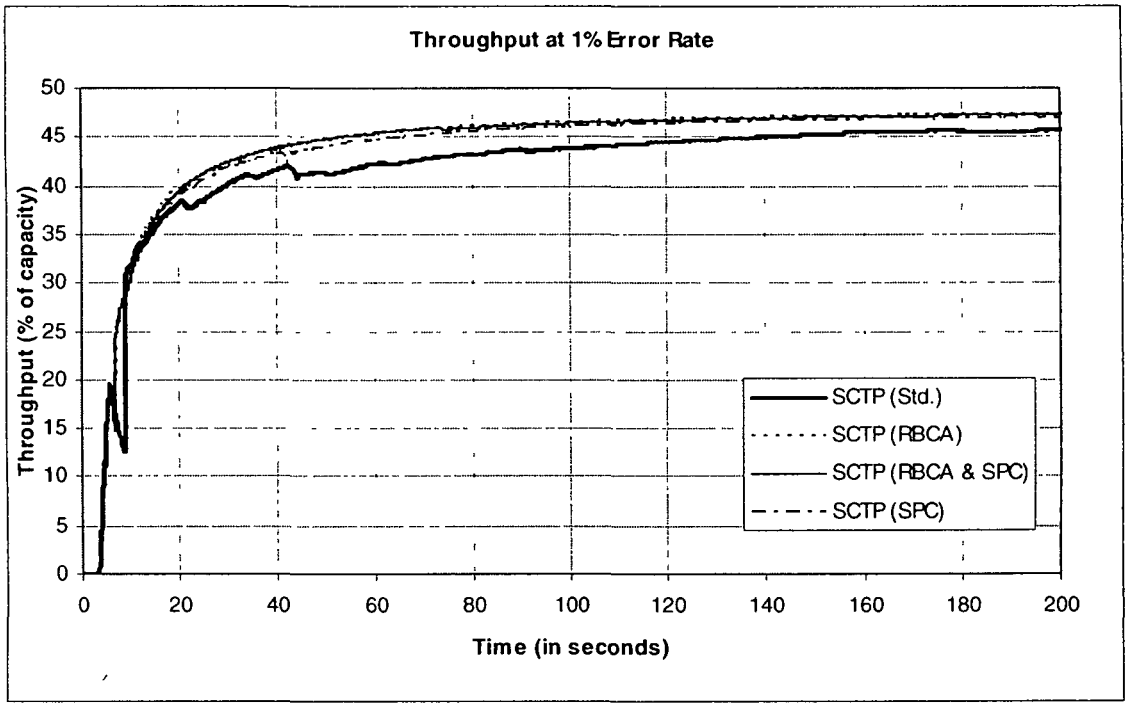

Fig. 6.10 Throughput comparison between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC), F=2.0 with 1% error

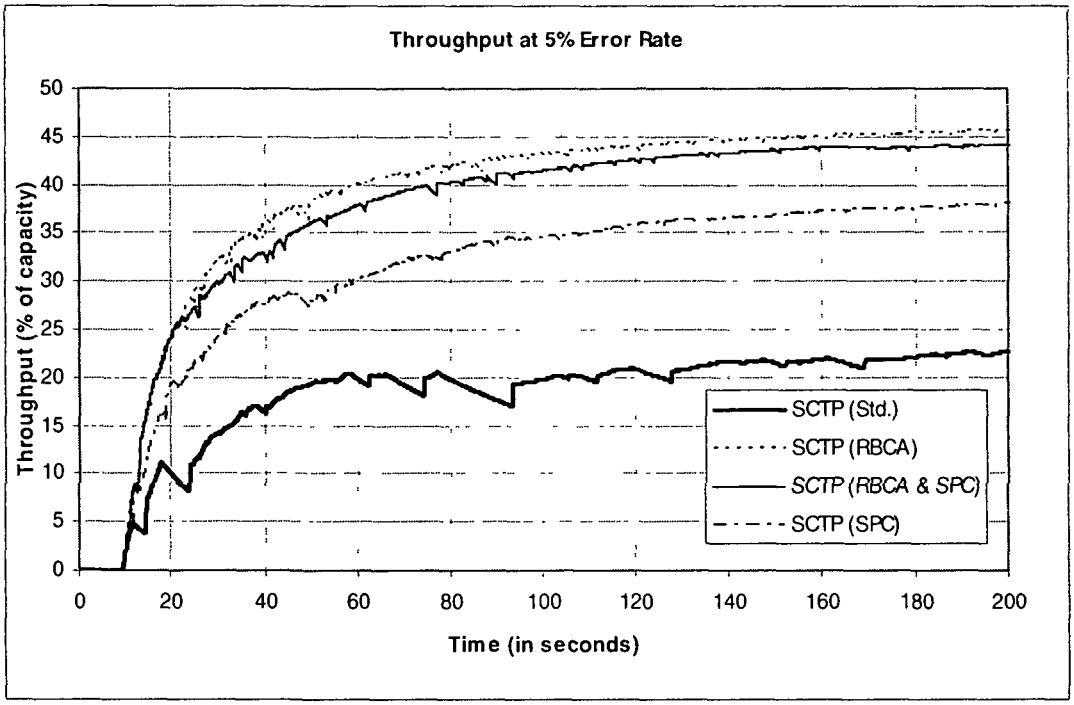

Fig. 6.11 Throughput comparison between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC), F=2.0 with 5% error

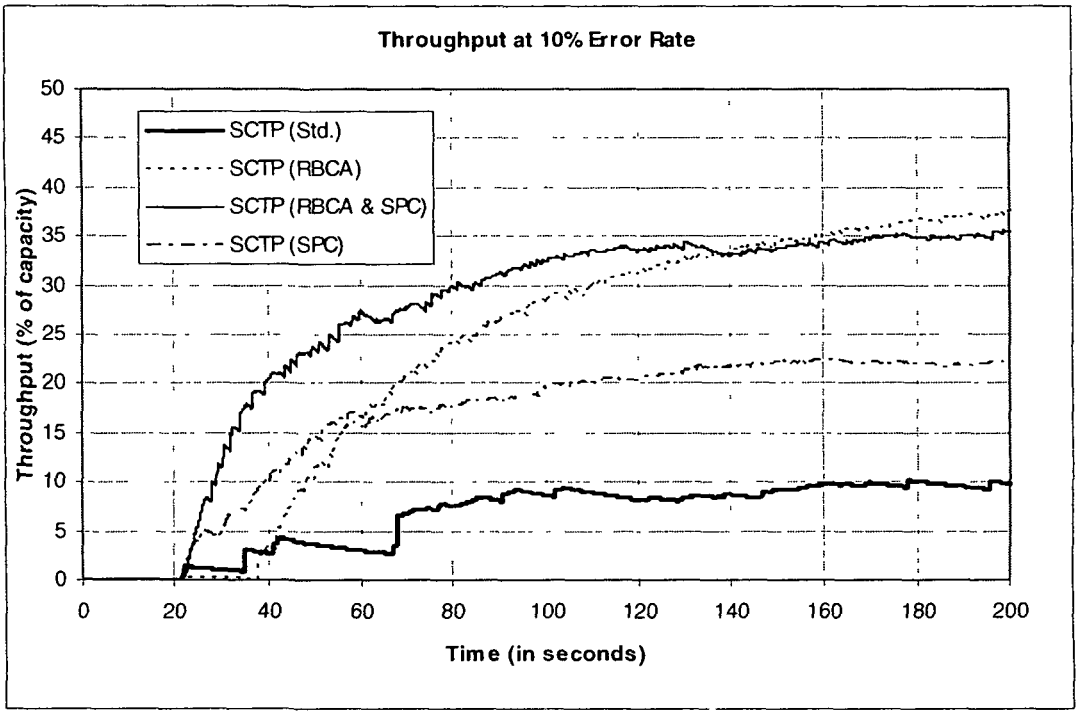

Fig. 6 12 Throughput comparison between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC), F=2.0 with 10% error.

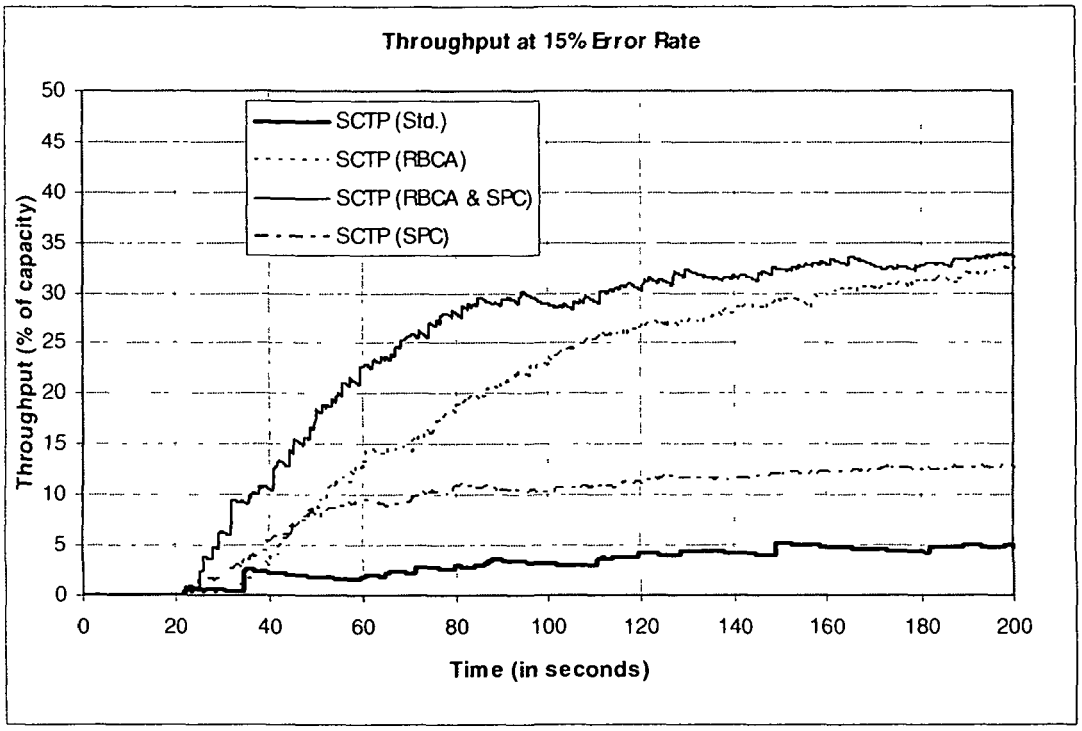

Fig. 6.13 Throughput comparison between SCTP (Std.), SCTP (RBCA), SCTP (RBCA & SPC) and SCTP (SPC), F=2.0 with 15% error.

It can be seen in Fig. 6.9 that at 0% error rate all the four versions of SCTP perform equally well. At error rate of 1% (Fig. 6.10) SCTP (Std.)'s performance drops a little more than the other three. At 5% error rate, as shown in Fig. 6.11, the difference opens up more and here SCTP (RBCA) and SCTP (RBCA & SPC) display throughput more than double of that of SCTP (Std.). The performance of SCTP (SPC) hangs in the middle. This trend continues for the higher error rates. Thus at high erroneous condition SCTP (Std.)'s performance degrades severely. It grossly underutilizes the available bandwidth under such conditions. This is due to the fact that SCTP (Std.) assumes all type of packet-losses as due to congestion. SCTP with RBCA shows far better performance than SCTP (Std.), since RBCA adapts the policy of handling congestion using RTT and making it independent of the packet-loss.

6.4 Summary

In this chapter, we have presented two schemes for congestion control in SCTP. The first scheme called RBCA is an adaptation of TCP RBCC. This adaptation is similar to that for wireless networks except that the delayed ACK mechanism of standard SCTP is not used here. This is done so that RTT can reflect the queuing delay more closely.

The second scheme called *Switch Path on Congestion (SPC)* exploits the multihoming feature of SCTP to avoid congestion. This scheme uses *RTT* to monitor the congestion situation in the primary path. When it detects a congestion situation approaching it switches to the alternate path and makes it the primary.

The two new schemes can be employed independently or together for congestion control in SCTP. These schemes have been implemented in NS2 simulation environment in different combinations for a comparative study of their performance. The results of the experiments indicate significant improvement in the performance of SCTP with the two new schemes in terms of reduced packet loss, lower packet latency and delay jitter, fairness, and utilization of network capacity. The new schemes also show much higher level of sturdiness in a noisy environment with high bit error rates. The experimental results as summarized in Table 6.1 below

Table 6.1

SCTP performance for various congestion control schemes in wired network environment for different traffic loads and error conditions

Scheme	*Packet-Drop (retransmiss-ion at source) (% of Injected packets)*	*Average Latency (Seconds)*	*Jitter (Std. Dev.)*	*Capacity Utilization*	*Fairness*	*Capacity Utilization in noisy environment (in %) with a single connection*
SCTP with RBCA	*0% (5)* *0% (25)* *(F=2.0)*	*0.1244 (5)* *(F=2.0)* *0.1427 (5)* *(F=2.3)*	*0.00995 (5)* *(F=2.0)* *0.02085 (5)* *(F=2.3)*	*49% (5)* *49% (25)* *(F=2.0)*	*0.99 (5)* *0.99 (25)* *(F=2.0)*	*45% (5% Err)* *37% (10% Err)* *(F=2.0)*
SCTP Standard	*2.82% (5)* *17.31%(25)*	*0.6002 (5)*	*0.09078 (5)*	*50% (5)* *59% (25)*	*0.99 (5)* *0.96 (25)* *(F=2.0)*	*22% (5% Err)* *10% (10% Err)*
SCTP with RBCAC & SPC	*0% (5)* *0.17% (25)* *(F=2.0)*	*0.1242 (5)* *(F=2.0)* *0.1424 (5)* *(F=2.3)*	*0.00992 (5)* *(F=2.0)* *0.01414 (5)* *(F=2.3)*	*92% (5)* *96% (25)* *(F=2.0)*	*0.93 (5)* *0.99 (25)* *(F=2.0)*	*44% (5% Err)* *35% (10% Err)* *(F=2.0)*
SCTP with SPC	*0.88% (5)* *4.10% (25)*	*0.1854(5)*	*0.08664(5)*	*82% (5)* *96% (25)* *(F=2.0)*	*0.99 (5)* *0.99 (25)* *(F=2.0)*	*38% (5% Err)* *22% (10% Err)*

Note:- 1. The numbers 5 and 25 in the bracket indicate the number of connections.

2. While considering capacity utilization sum of the capacities of the two paths is taken as the capacity.

Chapter 7

Conclusion

In this thesis we have studied the congestion control mechanisms in two reliable transport protocols, namely, TCP and SCTP. TCP being the most widely used transport protocol and SCTP being a promising protocol has generated lot of interest both in the research arena as well as the industry. We have analyzed causes of the well reported lacunae in the congestion control mechanism of the TCP protocol, namely-

1. TCP's driving the network into congestion causing long queuing delays, packet loss and retransmissions that lead to high packet latencies, large delay jitters, and overloading of the network with retransmission traffic.
2. Considering any packet loss to be due to congestion and thus making the protocol unsuitable for networks with wireless links, where packet losses due to causes other than congestion are very common.

Based on the analysis we came to the understanding that TCP's driving the network into deep congestion as a part of probing for estimating the available bandwidth for the connection is the cause of the two problems above. Thus it was concluded that both the above problems can be addressed if TCP's driving the network into deep congestion could be avoided by maintaining the queue occupancy in the routers at a lower level.

Considering this we have devised a new congestion control mechanism based on *RTT* called *RTT based congestion control (RBCC)* for TCP that avoids driving the network into deep congestion. This scheme maintains a *threshold RTT* for the connection and continuously adjusts the *congestion window* such that the *RTT* hovers around the *threshold RTT*. We have developed adaptations of this new algorithm for both wired networks as well as networks with wireless links. These adaptations referred to as *RBCC* and *RBCC-WL* respectively, have been implemented in NS2 environment

and tested for comparison of their performance with respect to existing prominent variants of TCP, namely, Newreno, Vegas, SACK, and Snoop. These two adaptations have been found to perform well in the following respects without any degradation in the throughput-

- The percentage of packets retransmitted drop drastically.
- The packet latencies drop significantly.
- There is substantial reduction in delay jitters.
- The fairness maintained is generally higher.
- The queue occupancy in the routers drop substantially.

In addition, it is also possible to have control over packet latency and delay jitter with a marginal tradeoff on throughput.

The congestion control mechanism of TCP is also inherited by SCTP. As a result the problems in TCP due to these mechanisms also migrate to SCTP. We have therefore developed an adaptation of the new congestion control scheme for SCTP too. This referred to as *RTT based congestion Avoidance (RBCA)* has been implemented in NS2 environment and tested for comparison of its performance with the standard SCTP that incorporates SACK and Delayed ACK as built-in mechanisms. The simulated experiments show similar improvements in the performance in SCTP with RBCA, in place of the existing mechanisms, as in the case of TCP.

It has been observed that the standard SCTP does not make full utilization of its multihoming feature. It maintains an alternate path but uses it only for the retransmission of lost packets. We have devised another new congestion control mechanism for SCTP referred to as *Switch Path on Congestion (SPC)* that makes a balanced use of both, the primary as well as the alternate path. This mechanism can be used in SCTP along with RBCA as well as with standard SCTP. Simulated experiments in NS2 environment show significant improvement in the performance of SCTP in both these cases in terms of packet loss, throughput, packet latency, and delay jitter. The new schemes also make SCTP much more robust to the high rate of bit errors prevalent in wireless environment.

7.1 Limitations and Scope for Future Work

Some of the limitations of the work presented in the thesis and that need further exploring are the following:

1. One of the assumptions of the *RTT Based Congestion Control* technique is that the components in *RTT* other than the queuing delay do not vary for a given path. This assumption is not correct for networks with wireless segments that use IEEE 802.11 MAC layer. Some mechanism needs to be developed to deal with the varying MAC layer delay in IEEE 802.11 based Wireless LANs.
2. In a dynamic network scenario the route to a destination may get changed during a TCP connection or an SCTP association. Such a change in the route is likely to cause change in the delay characteristics of the connection and make any previous computation of the *threshold RTT* invalid.
3. In high-speed networks, where the bandwidth is in the range of gigabits, the queuing delay component is likely to be a very small fraction of the *RTT*. So the change in the *RTT* due to the variation in the queuing delay is likely to be negligible. Therefore, adapting RBCC to high-speed networks will require some further study.
4. The Algorithm for Dynamic Computation of Threshold RTT has higher algorithmic complexities and has been tested only in wired network with one set of delay and bandwidth. So it requires further study to simplify and generalize this algorithm.
5. Delayed ACKs are not used in the new scheme for SCTP to let *RTT* reflect the queuing delays closely. This creates some otherwise avoidable ACK traffic towards the upstream. Adapting the scheme to the use of delayed ACKs shall help in avoiding this shortcoming.
6. One issue in case of the SCTP adaptation is that the two paths between the source and the sink may be of asymmetric delays and bandwidths. Studying the behavior of RBCA and SPC in such scenarios will be useful.
7. The SPC scheme for SCTP currently provides for only two alternate paths. It may be useful to generalize the scheme for more than two alternate paths.
8. In the present work no queuing theory based analysis has been made on the proposed schemes. A queuing theory based analysis for the schemes is therefore due for these.

Bibliography

[AA03] Anderson E. J., Anderson T. E., "On the Stability of Adaptive Routing in the Presence of Congestion Control", IEEE INFOCOM 2003.

[APS99] Allman M., Paxson V., Stevens W., "TCP Congestion Control", IETF, RFC 2581, April 1999.

[ASL03] Al A. A. El, Saadawi T., Lee M., "Load Sharing in Stream Control Transmission Protocol", Proceeding of the Internet and Multimedia Systems and Applications, 2003.

[BB95] Bakre A., Badrinath B., "I-TCP: Indirect TCP for Mobile Hosts", In Proceedings of IEEE ICDCS'95, 1995; 136-143.

[BF05] Basto V., Freitas V., "SCTP Extension for Time Sensitive Traffic", 5th International Network Conference, Somos Island, Greece, July 5-7, 2005.

[BKVP96] Bakshi B. S., Krishna P., Vaidya N. H., Pradhan D. K., "Improving Performance of TCP over Wireless Networks", Technical Report 96-014, Texas A & M University, 1996. (EBSN).

[BOP94] Brakmo Lawrence S., O'Malley Sean W. and Peterson Larry L., "TCP Vegas: New Techniques for Congestion Detection and Avoidance", Department of Computer Science, University of Arizona, August 1994.

[BPSK97] Balakrishna H., Padmanabhan V. N., Seshan S., Katz R. H., "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", December 1997.

[Bra89] Braden R. T., "Requirements for Internet Hosts – Communication Layers", RFC 1122, IETF, October 1989.

[BRHG04] Bitorika A., Robin M., Huggard M., Goldrick C. M., "A Comparative Study of Active Queue Management Schemes", ICC 2004.

[BS97] Brown K., Singh S., "M-TCP: TCP for Mobile Cellular Networks", In Proceedings of the ACM SIGCOMM Computer Communication Review. 1997; 19-43.

[BSAK95] Balakrishna H., Seshan S., Amir E., Katz R., "Improving TCP/IP Performance over Wireless Networks", In Proceedings of the 1st ACM International

Conference on Mobile Computing and Networking (Mobicom), November 1995. (SNOOP).

[BV98] Biaz S., Vaidya N. H., "Sender-Based Heuristics for Distinguishing Congestion Losses from Wireless Transmission Losses", Technical Report, #98-013, Department of Computer Science, Texas A&M University, June 1998.

[CAS03] Caro A. L. Jr., Amer P. D. and Stewart R., "Retransmission policies with transport layer multihoming", ICON '03, Sydney. Australia, September 2003.

[CBDA98] Chaher N., Barakat C., Dabbous W., Altman E., "Improving TCP over Geostationary Satellite Links", Rapport de recherché no. 3573, INRIA Sophia Antipolis, France, December 1998.

[CGG04] Casetti C., Greco R., Galante G., "Load balancing over multipaths using bandwidth-aware source scheduling", 7th International Symposium on Wireless Personal Multimedia Communications, WPMC~2004, Abano Terme, Italy, September 12--15, 2004.

[CIALHS03] Caro A. L. Jr., Iyengar J. R., Amer P. D., Ladha S., Heinz G. J. II, Shah K. C. "SCTP: A proposed Standard for Robust Internet Data Transport", IEEE Computer Society, November 2003.

[CJ89] Chiu D.M., Jain R., "Analysis of the Increase and Decrease Algorithm for Congestion Avoidance in Computer Network", North-Holland, Computer Networks and ISDN Systems 17 (1989) 1-14

[Com03] Comer D. E., "Internetworking with TCP/IP Principles, Protocols and Architecture", 4th Edition, Pearson Education, 2003.

[DMT96] Durst Robert C., Miller Gregory J. and Travis Eric J., "TCP Extension for Space Communications", ACM Mobicom, November 1996.

[DS04] Dahal M., Saikia D. K. "An RTT Based Congestion Control Scheme for TCP and its Performance in Wireless Environment", Proceedings of the National Workshop on Wireless Network and Mobile Computing (WNMC'04), Aug 26-28, 2004, Tezpur University, Assam, India.

[DS06I] Dahal M., Saikia D. K. "A Comparative Study of RTT Based Congestion Control (RBCC) Scheme with TCP SACK in Wireless Environment", Networks, Data Mining and Artificial Intelligence (Trends and Future Directions), Narosa Publishing House, New Delhi, India, January 2006:13-22.

[DS06II] Dahal M., Saikia D. K. "RTT Based Congestion Control and Path Switching Scheme for SCTP", ICCT 2006: Proceedings of the 10th International Conference on Communication Technology, November 27-30 2006, Guilin, China.

[FA04] Fu S., Atiquzzaman M. "SCTP: State of the Art in Research, Products, and Technical Challenges", IEEE Computer Magazine, April 2004.

[FH98] Ferguson P., Huston G., "Quality of Service on the Internet: Fact, Fiction, or Compromise? ",http://www.potaroo.net/papers/inet98/index.html.

[FH99] Floyd S., Henderson T., "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 2582, April 1999.

[FIOJ05] Funasaka J., Ishida K., Obata H., Jutori Y., "A Study on Primary Path Switching Strategy of SCTP", In the Proceedings of 7th International Symposium on Autonomous Decentralized Systems, Chengdu, China, 4-6 April 2005.

[FJ93] Floyd S., Jacobson V., "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transaction on Networking, Vol. 14, pp. 397-413, 1993.

[Flo00] Floyd S., "Congestion Control Principles", RFC 2914, September 2000.

[For06] Forouzan B. A., "TCP/IP Protocol Suit", Tata McGraw-Hill, 2006.

[FRT02] Fortz B., Rexford J., Thorup M., "Traffic Engineering with traditional IP Routing Protocols", IEEE Communication Magazine, October 2002.

[Fri46] Friis H. T., A note on a simple transmission formula, Proc. IRE, 34, 1946.

[FV00] Fall K., Varadhan K., "The *ns* Manual", The VINT Project, A Collaboration between UC Berkeley, LBL, USC/ISI, and Xerox PARC, December 2000.

[GCKB01] Gevros P., Crowcroft J., Kirstein P., Bhatti S., "Congestion Control Mechanisms and the Best Effort Service Model", IEEE Network, May-June 2001.

[GRL99] Griffith T. T., Rafelghem S. J. V., Legare D. J., "Implementing TCP Extensions for Error-Prone Satellite Communications", American Institute of Aeronautics and Astronautics, 1999.

[HK99A] Henderson T. R., Katz R. H., "TCP Performance over Satellite Channels", Report No. UCB/CSD-99-1083, Computer Science Division, University of California, Berkeley, December 1999.

[HK99B] Henderson T. R., Katz R. H, "Transport Protocol for Internet-Compatible Satellite Networks", IEEE Journal on Selected Areas of Communications, 1999.

[Hoe95] Hoe J., "Startup Dynamics of TCP's Congestion Control and Avoidance Schemes", Master's Thesis, MIT, 1995. URL "http://ana-www.lcs.mit.edu/anaweb/ pdf-papers/ hoe-thesis.pdf".

[IEEE99] Information Technology - Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11-1999, 1999.

[IOM04] Ishtiaq A., Okabe Y., Kanazawa M., "Modified Congestion Control of SCTP over Wide Area Networks", Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004).

[ISAS03] Iyenger J. R., Shah K. C., Amer P. D., Stewart R., "Concurrent Multipath Transfer Using SCTP Multi-homing", Tech Report 2004-02, CISC Dept, Univ of Delaware, 2003.

[Jac88] Jacobson V., "Congestion Avoidance and Control", ACM Computer Communication Review, Vol. 18, No. 4, August 1988, pp. 314-29

[Jac90] Jacobson V. "Modified TCP Congestion Avoidance Algorithm", Technical Report, April 30 1990. Message to end2end-interest mailing list, April 1990, URL "ftp://ftp.ee.lbl.gov/email /vanj.90apr30.txt".

[Jai89] Jain R., "A Delay Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks", ACM CCR, 19:56-71, 1989.

[JWL04] Jin C., Wei D. X., Low S.H., "FAST TCP:Motivation, Architecture, Algorithms, Performance", In the Proceedings of IEEE Infocom, Hong Kong, March 2004.

[JWL03] Jin C. et el, "TCP FAST: From Theory to Experiments", http://netlab.caltech.edu/FAST, December 2003.

[KHR02] Katabi D., Handley M., Rohrs C., "Congestion Control for High Bandwidth-Delay Product Networks", SIGCOMM'02, Pennsylvania, USA, August 2002.

[KMPM04] Kelly A., Muntean G., Perry P., Murphy J., "Delay-Centric Handover in SCTP over WLAN", Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE, Vol. 49 (63), 2004, ISSN 1224-600X

[KNKK04] Kashihara S., Nishiyama T., Koga H., Kadobayashi Y., Yamaguchi S., "Path Selection Using Active Measurement in Multi-home Wireless Networks", In the proceedings of International Symposium on Applications and the Internet, 2004.

[Lei03] Leiner B. M. etl el., "A Brief History of Internet", Internet Society, http://www.isoc.org/internet/history/brief.shtml, Version 3.32, December 2003

[Liu92] Liu H., "Traffic Shaping for Congestion Control in High Speed ATM Networks", MS Thesis, College of Graduate Studies and Research, University of Saskatchewan, Canada, August 1992.

[LPPBAB06] Lu Y., Pan R., Prabhakar B., Bergamasco D., Alaria V., Baldini A., "Congestion control in networks with no congestion drops.", Allerton, September 2006, www.ieee802.org/1/files/public/docs2006/au-Lu-et-al-BCN-study.pdf

[MCGLS00] Mascolo S., Casetti C., Gerla M., Lee S. S., Sanadidi M., "TCP Westwood: congestion control with faster recovery", UCLA CSD Technical Report #200017, 2000.

[MM96] Mathis M., Mahdavi J., "Forward Acknowledgement: Refining TCP Congestion Control", Proceedings of ACM SIGCOMM'96, pp. 281-291, Stanford, CA, August 1996.

[MMFR96] Mathis M., Mahdavi J., Floyd S., Romanow A. "TCP Selective Acknowledgment Options" RFC 2018, October 1996.

[MNR00] Martin J., Nilsson A., Rhee I., "The Incremental Deployability of RTT-Based Congestion Avoidance for High Speed Internet Connections", SIGMETRICS 2000.

[MNR03] Martin J., Nilsson A., Rhee I., "Delay-Based Congestion Avoidance for TCP", IEEE/ACM Transactions on Networking, Vol. 11, No. 3, June 2003.

[MTZ05] Mamatas L., Tsaoussidis V., Zhang C., "BOTTLENECK-QUEUE BEHAVIOR: How much can TCP know about it?", Proceedings of the 5th International Network Conference (INC), Samos, Greece, July 2005, http://utopia.duth.gr/~emamatas/inc2005.pdf.

[Nag84] Nagle J., "Congestion Control in IP/TCP Internetworks", RFC 896, IETF, January 1984.

[NS2] NS2: The UCB/LBNL/VINT Network Simulator (NS), URL: "http::://www.isi.edu/nsman/ ns/".

[PB94] Perkins Charles E., Bhagwat Praveen, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers" ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications, Pages 234-244, August 1994

[PJD04] Prasad R. S., Jain M., Dovrolis C., "On the Effectiveness of Delay-Based Congestion Avoidance", Georgia Tech, http://dsd.lbl.gov/DIDC/PFLDnet2004/papers/Dovrolis.pdf.

[PLPC04] Park E. C., Lim H., Park K. J., Choi C. H., "Analysis and design of the virtual rate control algorithm for stabilizing queues in TCP networks", Elsevier Computer Science, Computer Networks 44 (2004) 17-41.

[Pos81] Postal J., "Transmission Control Protocol", RFC 793, September 1981.

[Rap03] Rappaport T. S., "Wireless Communications, Principles and Practice", Pearson Education, 2003

[Ray99] Ray G., "Quality of Service in Data Networks: Products", http://www.cse.wustl. edu/~jain/cis788-99/ftp/qos_products/index.html.

[RF99] Ramakirshna K. and Floyd S., "A proposal to add Explicit Congestion Notification (ECN) to IP", RFC 2481, January 1999.

[RL06] Ribeiro E. P., Leung V. C. M., "Minimum Delay Path Selection in Multi-Homed Systems with Path Asymmetry", IEEE Communication Letters, Vol. 10, No.3, March 2006.

[RM98] Ratnam K., Matta I., "WTCP: An Efficient Mechanism for Improving TCP Performance over Wireless Links", In the Proceedings of the 3rd IEEE Symposium on Computer and Communications (ISCC'98), June 1998.

[RRQ03] Ryu S., Rump C., Qiao C., "Advances in Internet Congestion Control", IEEE Communications Surveys & Tutorials, Third Quarter, 2003.

[SD03] Saikia D. K., Dahal M. "Packet Loss Free Congestion Control in TCP for Controlled Packet Latency and Optimal Throughput", IEEE TENCON-2003: Proceedings of the Conference on Convergent Technologies for the Asia Pacific, October 15-17 2003, Bangalore, India.

[SP98] Schelen O., Pink S., "Aggregate Resource Reservation over Multiple Routing Domain", In the proceedings of IFIP Sixth International Conference on QoS, California, May 1998.

[Ste00] Stewart R. et al., "Stream Control Transmission Protocol", IETF RFC 2960, October 2000.

[Ste01] Stevens W. R., TCP/IP Illustrated, Vol.1, Addison-Wesley Professional Computing Series, 2001.

[STJ02] Santos J. R., Turner Y., Janakiraman G., "End-to-End Congestion Control for System Area Networks", Internet Systems and Storage Laboratory, HP Laboratories Palo Alto, HPL-2002-4 (R.1), May 2002

[Tan99] Tanenbum S., Computer Networks, 4th Edition, Section 5.3: Congestion Control Algorithms, Prentice Hall, September 1999.

[TM02] Tsaoussidis V., Matta I., "Open Issues on TCP for mobile computing", Wireless Communication and Mobile Computing, 2002; 2:3-20.

[VB99] Vaidya N. H.. Biaz S., "Is the Round-trip Time Correlated with the Number of Packets in Flight?", Department of Computer Science, Texas A & M University, Technical Report, 1999.

[VSK06] Vayias E., Soldatos J., Kormentzas G., "Traffic Shaping based on an Exponential Token Bucket for Quantitative QoS: Implementation and Experimentation on DiffServ Routers", www.ait.gr/research/RG1/files/ CCJ_06b.pdf.

[WC91] Wang Z., Crowcroft J., "A New Congestion Control Scheme : Slow Start and Search (Tri-S)", ACM Computer Communication Review, Vol. 21, No. 1, January 1991

[WC92] Wang Z., Crowcroft J., "Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm", ACM Computer Communication Review, April 1992.

[XQYZ04] Xie H., Qiu L., Yang Y. R., Zhang Y., "On Self Adaptive Routing in Dynamic Environments- An Evaluation and Design using a Simple Probabilistic Scheme", Proceedings of 12th International Conference on Network Protocols (ICNP 2004), Berlin, Germany, October 2004.

[YSL03] Ye G., Saadawi T., Lee M., "Using Explicit Congestion Notification in Stream Control Transmission Protocol in Lossy Networks", P.704, 23rd International Conference on Distributed Computing Systems Workshop, 2003.

[ZT03] Zhang C., Tsaoussidis V., "Improving TCP Smoothness by Synchronized and Measurement-based Congestion Avoidance", NOSSDAV'03, Monterey, California, USA, June 1-3, 2003.

Appendix I

Glossary

ACK	An Acknowledgement packet sent by receiver to the sender on reception of a data-packet.
ARP	Address Resolution Protocol is used for translating IP Address to Hardware Address.
ARQ	Automatic Repeat Request
BER	Bit Error Rate, which is the cause of packet-loss in Wireless Network.
BS	Base Station in a Infrastructure based Wireless Network.
Chunk	A unit of data or control information used in SCTP. A SCTP data-packet may contain several data or control chunks.
CMT	Concurrent Multi-path Transfer scheme for data transfer over multiple paths in SCTP.
Congestion	The load of data-packets on a transmission path that exceed its capacity.
CRC	Cyclic Redundancy Check
CTS	Clear-To-Send
cwnd	Congestion Window variable used in TCP and SCTP for congestion control.
cwndIncr	The value by which *cwnd* size is increased or decreased.
DCA	Delay-based Congestion Avoidance technique.
Delay	The time consumed by a data-packet while being transmitted on a path towards the destination which may be due to processing, propagation or waiting.
Delay-Jitter	The variation in packet latency is known as delay-jitter.
DoS	Denial of Service

Drop Tail	A queuing mechanism which is used in routers that drops the packet from the tail of the queue. It is also known as Tail Drop.
Fairness	The degree uniformity in sharing of the network bandwidth by a number of competing flows.
FEC	Forward Error Correction
FH	Fixed Host
FIFO	First in first out, this is a queuing mechanism that serves the first incoming packet first in the queue.
HOL	Head-of-Line
HowOld	A variable to keep track of how old a path is in SCTP. A lesser value implies the path being used more recently.
ICMP	Internet Control Message Protocol which is used for transmitting control and error messages by IP.
IDT	Internet Data Transport
IP	Internet Protocol is the network layer in the Internet that is used for forwarding data-packets.
ISO	International Organization of Standards
LAN	Local Area Network
MAC	Medium Access Control
MH	Mobile Host
MSS	Maximum Segment Size.
MTU	Maximum Transfer Unit.
Newreno	A version of TCP that provides recovery in case of multiple packet losses in a single-window.
Non-Queuing Delay	Delay caused by other than queuing in router such as propagation, transmission, processing, retransmission and route change.
NS	Network Simulator, a standard software for simulating computer networks.
OldCwnd	The value of *cwnd* for a path prior to switching in SCTP.
OSI	Open System Interconnection architecture of ISO for

	Computer Networks.
Packet-Drop	When a router is too congested and can not hold anymore packets in its buffer, then subsequent arrival of packets are discarded. This is known as packet-drop.
Packet-Loss	When data-packets are sent by source, but are not correctly received by destination or not received at all due to packet drop by router or due to bit-errors over noisy link. This phenomenon is known as packet-loss.
Path Switching	A change in the transmission path from the current one to an alternate path available.
PLE	Path Loss Exponent; Used in Shadowing (Propagation) Model of Wireless Network for simulating obstruction.
PPRTT	The RTT prior to PRTT.
PRTT	The Previous RTT.
QoS	Quality of Service.
Queue Length	The length of buffer space in a router occupied by data-packets waiting to be forwarded towards the destination.
Queuing Delay	The delay suffered by a data-packet while waiting in the router queues.
RBCA	The RTT Based Congestion Avoidance scheme for SCTP.
RBCC	The RTT Based Congestion Control scheme for TCP.
RBCC-WL	The RTT Based Congestion Control for TCP adapted for Wireless Networks.
Router	A network node that receives packets from one network link and forwards it on another network link towards the destination.
RTS	Request-To-Send message used in wireless MAC layer protocol.
RTT	Round trip time. It is the time between the sending of a packet and the arrival of it's acknowledgement at the sender.
RTT_{min}	The minimum possible RTT for a given network path.

rwnd	Receiver Window in TCP and SCTP
SACK	Selective Acknowledgement, with the help of which TCP or SCTP can acknowledge a set of data packets together to the sender.
SCTP	The Stream Control Transmission Protocol is the reliable and connection-oriented transport protocol of Internet which has provision of multi-homing, multi-streaming and some security features.
Sink Node	One end of the communicating nodes that receives data-packets from the source node. It is also known as destination node.
Source Node	One end of the communicating nodes that sends the data-packets to the sink node.
SPC	Switch Path on Congestion. A congestion control scheme for SCTP that has multiple paths to a destination node.
SSN	Stream Sequence Number given to a SCTP Stream.
ssthresh	Slow-start threshold variable used in TCP and SCTP.
SYN	Synchronize TCP segment that is used for initiating a TCP connection by setting SYN flag.
TCP	Transmission Control Protocol is the reliable and connection-oriented transport protocol the Internet.
Threshold RTT	A value of RTT lying in the range between it's minimum and maximum values.
Throughput	The number of data-bytes received at the destination node in a given period of time.
TSN	Transmission Sequence Number assigned to an SCTP Packet.
UDP	User Datagram Protocol is the unreliable and connectionless transport protocol of the Internet.
Vegas	A version of TCP which uses RTT to calculate expected and actual throughput before deciding change in the congestion window.

WLAN	Wireless LAN
XCP	eXplicit Congestion control Protocol.