CONGESTION CONTROL ALGORITHMS OF TCP IN EMERGING NETWORKS

A Dissertation

by

SUMITHA BHANDARKAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2006

Major Subject: Computer Engineering

CONGESTION CONTROL ALGORITHMS OF TCP IN EMERGING NETWORKS

A Dissertation

by

SUMITHA BHANDARKAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	A. L. Narasimha Reddy
Committee Members,	Pierce E. Cantrell, Jr.
	Scott L. Miller
	Dmitri Loguinov
Head of Department,	Costas N. Georghiades

August 2006

Major Subject: Computer Engineering

ABSTRACT

Congestion Control Algorithms of TCP in Emerging Networks. (August 2006)
Sumitha Bhandarkar, B.E., Mysore University; M.S., Texas A&M University
Chair of Advisory Committee: Dr. A. L. Narasimha Reddy

In this dissertation we examine some of the challenges faced by the congestion control algorithms of TCP in emerging networks. We focus on three main issues. First, we propose TCP with delayed congestion response (TCP-DCR), for improving performance in the presence of non-congestion events. TCP-DCR delays the congestion response for a short interval of time τ, allowing local recovery mechanisms to handle the event, if possible. If at the end of the delay τ, the event persists, it is treated as congestion loss. We evaluate TCP-DCR through analysis and simulations. Results show significant performance improvements in the presence of non-congestion events with marginal impact in their absence. TCP-DCR maintains fairness with standard TCP variants that respond immediately.

Second, we propose Layered TCP (LTCP), which modifies a TCP flow to behave as a collection of virtual flows (or *layers*), to improve efficiency in high-speed networks. The number of layers is determined by dynamic network conditions. Convergence properties and RTT-unfairness are maintained similar to that of TCP. We provide the intuition and the design for the LTCP protocol and evaluation results based on both simulations and Linux implementation. Results show that LTCP is about an order of magnitude faster than TCP in utilizing high bandwidth links while maintaining promising convergence properties.

Third, we study the feasibility of employing congestion avoidance algorithms in TCP. We show that end-host based congestion prediction is more accurate than

previously characterized. However, uncertainties in congestion prediction may be unavoidable. To address these uncertainties, we propose an end-host based mechanism called Probabilistic Early Response TCP (PERT). PERT emulates the probabilistic response function of the router-based scheme RED/ECN in the congestion response function of the end-host. We show through extensive simulations that, similar to router-based RED/ECN, PERT provides fair bandwidth sharing with low queuing delays and negligible packet losses, without requiring the router support. It exhibits better characteristics than TCP-Vegas, the illustrative end-host scheme. PERT can also be used for emulating other router schemes. We illustrate this through preliminary results for emulating the router-based mechanism REM/ECN.

Finally, we show the interactions and benefits of combining the different proposed mechanisms.

To Hemanth

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Reddy, for giving me a chance to work with him. As a guide, mentor and critic he has helped me not only with my research but also to challenge myself and to push my boundaries. He allowed me to explore my interests freely both in my research as well as my career options, while ensuring that I did not stray too far away from the main objective. For all this and more, thank you, Dr. Reddy.

I would like to thank Dr. Cantrell, Dr. Loguinov and Dr. Miller for their interest in this research and for their invaluable suggestions. I would like to thank Dr. Vaidya for offering informative advice, every time I seeked his opinion.

The other students in my research group have been very helpful to me throughout the course of my study and I would like to express my deepest gratitude to them. Special mention goes to Phani Achantha, Saurabh Jain, Zilli Zhao, Sukwoo Kang and Yong Liu - they never hesitated to lend a patient ear or a helpful hand and were the best people one could work with.

I have always dreamt of pursuing a Ph.D. But without the constant support and encouragement from my husband, Hemanth, I doubt if it would ever be anything more than just a dream. He believed in me and stood by my side - rejoicing when things went well and gently nudging me along when things weren't quite so bright. He is my source of motivation and the force that keeps me going.

Finally, I would like to thank my parents - for giving me the love for learning and the encouragement to pursue my interests, my sisters - for their never-ending faith in me, and my in-laws - for going out of their way to understand, encourage and support.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

TCP has been the de-facto transport protocol for the Internet for over two decades. The scale of the Internet and its usage has increased by several orders of magnitudes. The nature of applications has significantly changed. Some of the assumptions made during the early design process are no longer valid. And yet, TCP remains the work horse of the TCP/IP protocol stack based on which the Internet runs. The reason TCP enjoys this center-stage is that it constantly evolves to keep up with the changing network demands. For instance, [1] lists over 75 RFCs[1] related to TCP that have been published as of early 2006. These range from standards documents to informational ones. The study presented in this dissertation is part of the continuous process to inspect some of the shortcomings of TCP in the current Internet and propose changes to keep it updated.

TCP resides in layer 4 of the 7-layer OSI network model. It provides a connection-oriented, reliable, byte-stream service that is both flow and congestion controlled to the upper layers (application layer), while assuming or expecting little from the lower layers (IP layer and below). This is accomplished by a complicated set of algorithms. In this study, we limit our focus to the congestion control algorithms of TCP.

The congestion control functionality of TCP is provided by four main algorithms namely slowstart, congestion avoidance, fast retransmit and fast recovery in conjunction with several different timers. Slowstart uses exponential window increase to

---

The journal model is *IEEE Transactions on Automatic Control*.

[1]RFCs (Request For Comments) are the official documents of the Internet Protocol suite that are defined, recorded and published by the Internet Engineering Task Force (IETF) and the Internet Engineering Steering Group (IESG) as part of the the Internet standards process.

quickly bring a newly starting flow to speed. In steady state, the flow mostly uses congestion avoidance in conjunction with fast retransmit/recovery. These algorithms implement the classic additive increase/multiplicative decrease of the congestion window. When no losses are observed, the congestion window is increased by one for the successful acknowledgment of one window of packets. Upon a packet loss, the window is decreased to half its earlier value, to clear out the bottleneck link buffers.

There are several challenges in current networks to this simple additive increase, multiplicative decrease policy. In this study we focus on three main issues: (a) loss of performance in the presence of non-negligible amount of non-congestion events (b) loss of performance in networks with high delay and bandwidth when multiplexing is low and (c) enhancing TCP to provide proactive congestion *avoidance*, instead of just reactive congestion *control*. The following sections provide a brief overview of the problems and why they motivate us. More detailed descriptions and our solutions to the problems are provided in the chapters that follow.

A. TCP in the Presence of Non-congestion Events

TCP is a self-sufficient protocol, in the sense that the sender uses information provided by the receiver in the form of acknowledgments, to determine the nature of congestion in the network. No explicit feedback is expected from the routers. This self-sufficiency is based on the assumption that anytime packets do not arrive at the receiver in the same order that the sender sent them, then it is due to congestion in the network. While in most conventional networks, this assumption is true, newer network environments challenge it. The most notable cases are when packets get reordered on high-speed switches or are lost due to channel errors in wireless networks. In these cases, the misclassification of the cause for out-of-order packet delivery or

packet losses as congestion, forces TCP to use multiplicative decrease of the congestion window and results in degraded performance. In Chapter II, we inspect this issue in detail and provide a general solution for improving the performance of TCP in the presence of non-negligible non-congestion events. This is followed by detailed evaluation of the solution for two specific cases - packet reordering and wireless channel errors. Since the solution proposes changes to the congestion control algorithms, we conduct comprehensive evaluation of the proposed scheme in "conventional" scenarios which do not contain any non-congestion events to show that it may be safe for deployment in the general Internet.

B. TCP on High BDP Links with Low Levels of Multiplexing

In the next part of the study, we focus on the performance of TCP in high bandwidth-delay links with bandwidth of the order of several hundreds of Mbps to a few Gbps and round trip time in the order of few milliseconds to few hundred milliseconds. Such links have historically been used only in the core of the network where several hundreds or thousands of flows get multiplexed. However, the increase in the installation of fiber links and the availability of low-cost high-speed network interface cards[2] and computer hardware/software that support it, has made it possible to use high Bandwidth-Delay product (BDP) links end-to-end. Currently, such links are used mainly between large research institutes and universities for sharing vast amounts of data. However with the advent of bandwidth-hungry latency-sensitive applications (e.g., on-demand video over IP), it is just a matter of time before the availability of high BDP links goes main stream. In such environments where few TCP flows share a high BDP link, the performance of TCP is poor due to the conservative choice of

[2]During the time that this dissertation is written, 1Gbps and 10Gbps network interface cards are easily available.

the parameters used in the additive increase, multiplicative decrease algorithms. In Chapter III, we study this problem in more detail and propose a solution for improving the performance of TCP in such high BDP environments. We study the behavior of the proposed solution, in comparison to other schemes that have been proposed, via analysis, simulation and Linux emulation.

C. Using TCP for Congestion Avoidance Instead of Congestion Control

Calling the AIMD algorithms used by TCP as "congestion avoidance" is a misnomer, since TCP does not try to "avoid" congestion proactively. Describing the algorithms used by TCP as "congestion control" is likely more appropriate, since the response of TCP is reactive to a perceived packet loss. The study of true congestion avoidance algorithms is as old as TCP itself. These algorithms use the information contained in the measured round trip time samples or the throughput to predict the onset of congestion, and respond before a packet loss occurs, in an effort to prevent the packet loss from occurring. Such proposals have come under attack in recent years from measurement studies that claim that the information in the delay signal is insufficient to accurately predict the congestion. In Chapter IV, we inspect the merits of these measurement studies. We investigate some of the means to improve the accuracy of the delay signal. While it may not be possible to entirely remove all the inaccuracies, we show that by choosing a proper early response function, we can alleviate the negative effects. Evaluation of the proposed scheme via extensive ns-2 simulations over wide ranging network parameters is also presented.

D. Putting It All Together

Finally in Chapter V, we investigate the benefits of combining all the proposed solutions together. We present results based on ns-2 simulations to show the benefits that can be achieved by using the solutions proposed in this work.

CHAPTER II

TCP IN THE PRESENCE OF NON-CONGESTION EVENTS

TCP was designed in the 1980's when the nature of the networks was very different from the nature of the network today. As a result, it makes some implicit assumptions, which may not be valid in todays' networks. For instance, one of the main assumptions driving the congestion control algorithms of TCP is that a packet loss implicitly signals congestion. Based on this assumption, when a packet loss is observed, the flow responds by multiplicatively reducing its window by half. When the reason for packet loss is indeed congestion, this helps flush the queue at the bottleneck link buffers [2]. But in cases where the packet loss is not due to congestion related reasons, or when a reordering event is falsely interpreted as a packet loss, this will result in an unnecessary window reduction and hence reduced link utilization.

Consider how the TCP flow identifies a packet loss. At the sender, every packet that is transmitted is given a unique monotonically increasing sequence number, which is embedded in the packet header. When the receiver receives a packet, it sends a short acknowledgment packet back to the sender. In the header of this packet, the receiver embeds the sequence number of the next packet it expects to receive. This will result in cumulatively acknowledging all the packets that have been received so far. Now suppose a packet is lost in transit. For all subsequent packets that the receiver receives, it repeatedly sends sequence number of the packet it expects to receive next, which is the packet that has been lost in transit. Since these acknowledgments repeat the same information, they are called "duplicate acknowledgments" (or dupacks for short). The sender has no definitive means to determine whether the packet that is indicated in the duplicate acknowledgments is indeed lost or just delayed (reordered). Hence, it waits until it receives three such dupacks, and then concludes that a packet

is lost [1].

Hence, the current implementations of TCP provides robustness against reordering, provided the reordered packet arrives at the receiver no later than three of its succeeding packets. If a packet is delayed in the network for longer and arrives at the receiver after four packets have been received with sequence numbers higher than its sequence number, then the above algorithm is unable to detect it and congestion response is triggered. Recent studies [3]-[4] have shown that while different parts of the Internet observe different extent of reordering, wait of three dupacks used in TCP may be an inappropriate heuristic. While this in itself is a good reason for investigating the robustness of TCP to packet reordering, the authors of [5] present a more compelling reason - the taboo against packet reordering prevents or restricts the research and deployment of several new, beneficial schemes on the Internet for providing efficient routing or differentiated services.

Next consider the case where a packet is indeed lost. In the absence of explicit notification, the sender cannot determine the cause for the loss. In the 1980's, the primary medium for communication was wired, and so the designers of TCP implicitly assumed that a loss indicated congestion. But today, the use of wireless networks is increasing at a tremendous rate. Wireless networks are characterized by higher channel error rates than wired networks. When TCP is used in wireless networks, the losses due to channel errors are mistaken for congestion losses and the sending rate is unnecessarily reduced, resulting in degraded performance [6].

These problems of TCP performance degradations have been studied earlier in their specific context. For instance [5, 7] propose solutions for improving the perfor-

[1] Another method for detecting loss is the expiration of the retransmission timer - but we limit our focus here to cases where packet losses are detected using duplicate acknowledgments.

mance of TCP when network paths have high levels of reordering and [8]-[9] focus on improving TCP performance in wireless networks. The solution provided in one specific context does not necessarily address the other contexts that cause the misinterpretation of congestion. In this chapter, we generalize the problem and provide a solution that improves the robustness of TCP to "non-congestion events". Non-congestion events are defined as events that are unrelated to the congestion in the network, but result in triggering duplicate acknowledgments.

A. TCP-DCR : The Proposed Solution

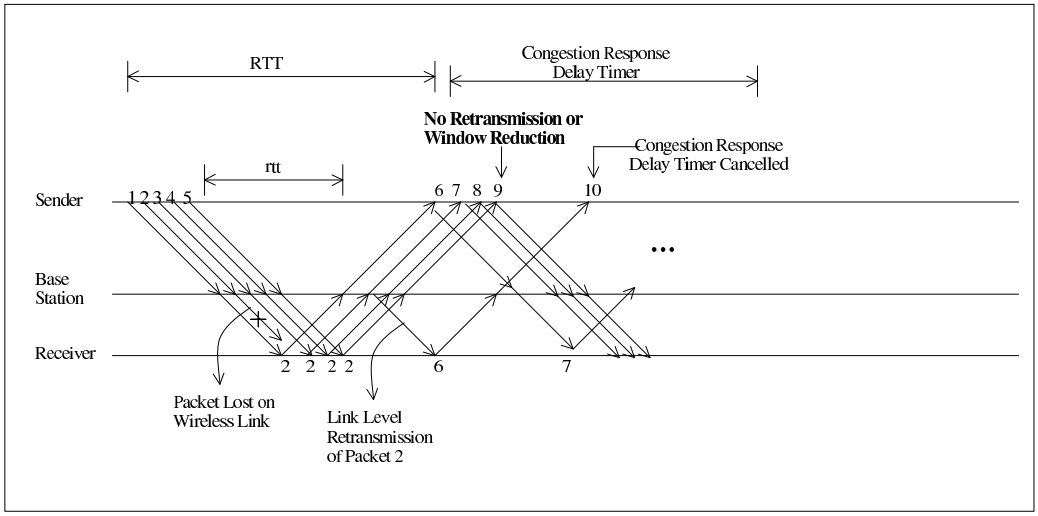
Our solution is intuitive and employs two simple ideas: (a) delay the congestion response of TCP for a short interval of time τ, creating room to handle any non-congestion events that may have occurred, and (b) employ "local recovery" techniques to recover from non-congestion events during this interval. If at the end of the delay τ, the event has not been handled, then it is treated as a congestion event. This simple concept fits into the general philosophy of segregation between the different layers of the network model. The modifications to TCP do not handle the non-congestion event, but rather, rely on some lower layer mechanism to do local recovery, if necessary. For instance, in case of packet reordering, no particular recovery mechanism is needed. If the delay τ in responding to congestion is chosen longer than the time the packet is delayed compared to its succeeding packets, then congestion control algorithms are not triggered. In case of wireless networks, simple modifications may be made to the link layer for recovering packet losses locally within the delay τ, when channel errors occur. Again, this will prevent the triggering of window reduction at the sender. Robustness to other non-congestion events (e.g. those related to mobility and handoff, etc) could also be potentially provided using this solution, provided,

that local recovery mechanisms are deployed for recovering from the non-congestion event within the time duration τ. To distinguish this flavor of TCP from the original, we call it the Delayed Congestion Response TCP (TCP-DCR for short).

1. Behavior of TCP-DCR

Fig. 1 shows the graphical representation of TCP-DCR when (a) the loss of a packet is due to transmission errors and (b) the loss of a packet is due to congestion. The TCP-DCR sender sends packets 1 through 5. However, due to channel error, say, packet 2 is lost. This is communicated by the link layer to the base station, say, by a negative acknowledgment (NACK). The base station immediately retransmits packet 2. But before packet 2 is recovered by link level retransmission, the TCP receiver sends dupacks for packet 2. In the case of the traditional implementations of TCP, three dupacks would trigger an immediate retransmission of packet 2 at the TCP sender, followed by an unnecessary window reduction. However, in the case of TCP-DCR, a delayed response timer of τ is started at the sender when the first dupack is received. During this delay period, packet 2 is recovered via link level retransmission causing the TCP receiver to generate a cumulative acknowledgment acknowledging packet 2. On the receipt of this acknowledgment, the TCP-DCR sender cancels the delayed response timer, and the unnecessary retransmission of packet 2 and reduction in congestion window is avoided. Also, TCP-DCR sends one new packet on the receipt of each dupack, if allowed by the congestion window, similar to the proposed standard "Limited Transmit Algorithm" [10]. This ensures that during the delay τ the sending rate of the TCP-DCR is the same as it was when the first dupack arrived.

In the case of a congestion loss, the packet cannot be recovered through link level retransmission. Upon the receipt of the first dupack the delayed response timer is started. However, since the packet is dropped by an intermediate router due to



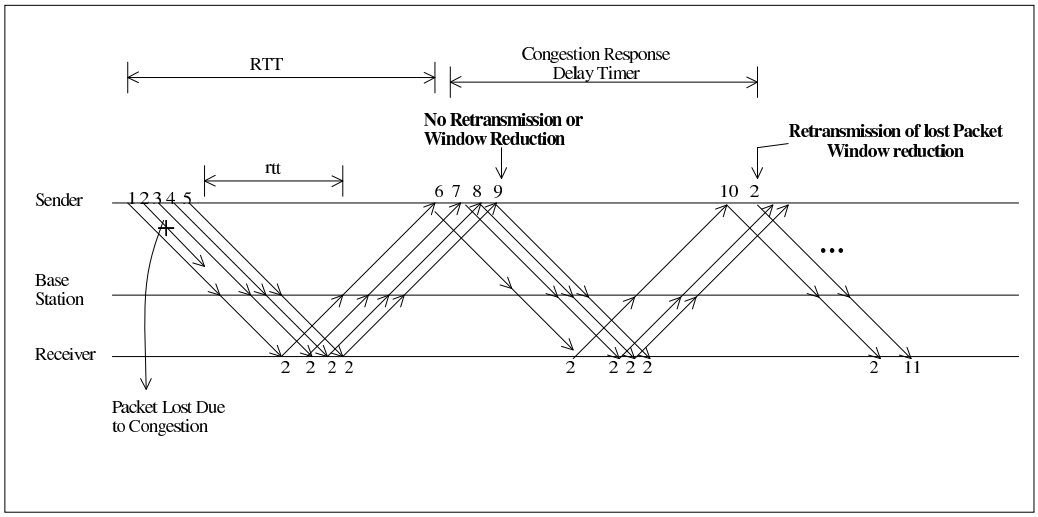


Fig. 1. Behavior of TCP-DCR

congestion, a cumulative acknowledgment for the lost packet is not received. When the timer expires, packet 2 is retransmitted and the congestion window is reduced to half. An important fact to remember here is that, the delay of τ does not cause the TCP-DCR sender to dramatically over-send packets because the protocol is still ACK-clocked. That is, a new packet is sent only upon the receipt of a dupack and the sending rate during the delay period is at most the sending rate when the first dupack arrived.

2. Choice of τ

The delay in responding to congestion determines the performance of TCP-DCR and the choice of τ is a critical aspect for the TCP-DCR modifications. Too large a delay would mean that the protocol responds too sluggishly to congestion in the network. Too small a delay would not allow the link layer sufficient time to recover from the losses due to channel errors. Hence it is essentially a tradeoff between unnecessarily inferring congestion, and unnecessarily waiting for a long time before retransmitting a lost packet and choosing the correct value for the delay is extremely important. In this section we provide guidelines for choosing reasonable bounds on the delay to make it useful, without adversely modifying the TCP behavior. At this point, we note that the current practice of waiting for three dupacks at the sender is merely a heuristic.

Consider first the wireless scenario. Fig. 2 shows a general case where the TCP receiver is connected to a base station over a wireless link. The wired path between the base station and the TCP sender could consist of several hops, but would not affect the discussion here and so is shown as a single hop. The round trip time between the base station and wireless link is indicated by rtt and the end-to-end round trip time between the TCP sender and the TCP receiver is indicated by RTT.

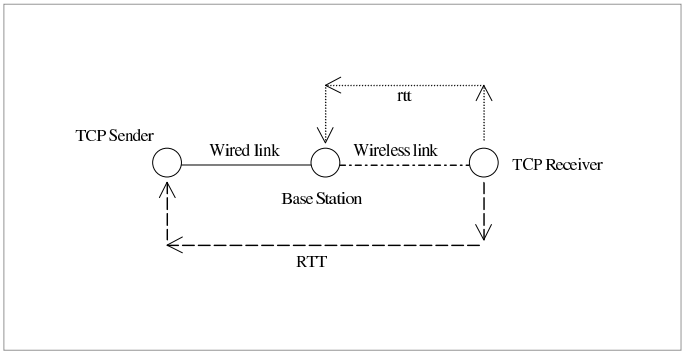


Fig. 2. Analysis of TCP-DCR in a Wireless Network with No Losses due to Congestion

In the above scenario, if we ignore ambient delays (e.g., inter-packet delay, queuing delay, etc.), a packet sent by the TCP sender at some time $t0$ reaches the base station at $t0 + (RTT/2 - rtt/2)$ and the receiver at time $t0 + RTT/2$. Suppose, a packet k sent at time $t0$ is lost on the wireless link due to channel errors. Then at $t0 + RTT/2 + rtt/2$ the base station receives indication that the packet k is lost. If it immediately retransmits the packet, then the packet k is recovered at the receiver at time $t0 + RTT/2 + rtt$. The sender receives an acknowledgment for the packet k at $t0 + RTT/2 + rtt + RTT/2$. Hence the sender would have to delay the congestion response by at least rtt time units after receiving three Dupacks, to allow the link layer to recover the packet. In practice, the inter-packet delays are non-zero and the TCP sender may not know the value of rtt. Hence, a simple solution would be to set the lower bound on the delay in congestion response to one RTT.

The TCP protocol uses two mechanisms for identifying congestion in the network - the receipt of three dupacks and the retransmission timeout (RTO). The receipt of three dupacks is considered to be an indication of mild congestion in the network and hence the response to it is the triggering of fast retransmission/recovery algorithms. An RTO, on the other hand is treated as an indication of severe congestion in the

network, and so in response to it, the congestion window is reset to 1 and the window evolution starts over with a slowstart. This is an extremely expensive operation. The choice of τ should be such that unnecessary retransmission timeouts are avoided. Thus, the upper bound on the delay τ is imposed by the retransmission timer of TCP. The RTO is usually set to RTT + 4 times the measured variance in RTT. The standard recommends a minimum of 1 second for the RTO, but many TCP implementations have a much smaller minimum, e.g., 100 ms. A choice of one RTT or less for the delay τ, can ensure that RTO can be avoided. Thus, the upper limit on the value of τ is one RTT.

In the case of packet reordering, the amount by which the packet is reordered could be highly variable - the time to recover the lost packet is the time that the reordered packet takes to reach the receiver. Hence there is no preset lower bound for the delay τ, that will facilitate the recovery of all reordered packets. However, the upper bound is still decided by the discussion above. So, a value of one RTT for tau is still a reasonable choice.

Based on the discussion above, we conclude that a choice of waiting for one RTT after the first dupack before responding to congestion is optimal. By setting the delay to one RTT, rather than a fixed value, we also provide inherent robustness to fluctuations in the queuing delays ensuring that we do not get into RTO timeout even during sudden changes in the network load.

3. Implementation

The TCP-DCR modifications need to be applied only to the sender. The receiver remains unmodified. The congestion response is delayed only during the congestion avoidance phase and hence does not modify the behavior during the slow start phase. During the congestion response delay, the congestion window continues to evolve as

indicated by the congestion avoidance algorithm (additive increase). However, only one new packet is transmitted in response to each dupack received. This is similar to the proposed standard limited transmit algorithm [10]. This ensures that TCP-DCR remains NACK-clocked during the congestion response delay period and a new packet is put on the network only when indication is received that one of the previously sent packets has left the network. Thus the sending rate of the TCP-DCR sender during τ remains at best, the same as when the first dupack was received.

If the congestion response delay timer expires, the fast retransmit/recovery algorithms are triggered. The *ssthresh* and the congestion window are set to half the current value of the congestion window just as it would be in a traditional implementation of TCP.

The sender can implement the delay either by using a timer or by modifying the threshold on the number of dupacks to be received before triggering the congestion recovery algorithms (*dupthresh*). The timer based implementation is quite straight forward, but depends on the clock granularity. In the dupack-based delay implementation, the sender could delay responding to congestion for a window of packets, with the window corresponding to the delay required. Thus, when τ is chosen to be one RTT, the sender would wait for the receipt of W dupacks, before responding to congestion, where W is the sending window when the first dupack is received. The implementation of the delay should take care that a faulty implementation does not result in an RTO. So, for the timer-implementation we suggest that the timer be set to one RTT as indicated by the *smoothed_RTT* estimate since the RTO estimate is computed based on the *smoothed_RTT*. In case of the dupack-based implementation, the number of dupacks correspond to the estimate of *current_instantaneous_RTT* and so we suggest that the new value for *dupthresh* be scaled by the factor $(smoothed\_RTT)/(current\_instantaneous\_RTT)$.

The TCP-DCR modifications work with most flavors of the TCP protocol. However, in this study we advocate the use of TCP-DCR with TCP-SACK to ensure that the performance can be maintained high even under the conditions of multiple losses per round trip time. When used with TCP-SACK, the only thing modified by TCP-DCR is the time at which the fast retransmit/recovery algorithm is triggered in response to dupacks generated by the first loss within a window of packets. All subsequent losses within the same window (irrespective of whether they are due to congestion losses or non-congestion events) are handled in exactly the same way as TCP-SACK would in the absence of TCP-DCR modifications. If the receiver is not SACK-capable, however, then the sender will have to use TCP-DCR with other flavors such as NewReno. If several packets are lost in one RTT, then the number of dupacks being received is less, and because of the NACK-clocked nature of the sender, it implicitly forces the sender the reduce its sending rate.

Use of delayed_acks will not intervene with the TCP-DCR modifications, provided that the implementation of delayed acks follow the guidelines in [11] that the dupacks (or SACKs) are not delayed.

4. Receiver Buffer Requirement When TCP-DCR Is Used

When TCP-DCR is used, the receiver will need to have additional buffer space to accommodate the extra packets corresponding to the delay τ, when a packet is lost due to congestion. Having these extra buffers allows TCP-DCR to achieve the best performance. However, if the buffers are not available, it does not degrade the performance drastically, but the maximum performance improvement is not achieved. This is because, apart from congestion control, TCP also provides flow control such that a faster sender does not flood a slow receiver. The flow control is achieved by using a receiver advertised window, such that at any point the TCP sender may not send

more packets than that allowed by $min(cwnd, rwnd)$ where $cwnd$ is the congestion window and $rwnd$ is the receiver advertised window. When the buffer space is not available, the receiver advertised window is small. As a result, during the delay τ even though the limited transmit and congestion window allow a packet to be transmitted it will not be sent if the $rwnd$ (and hence the receiver buffer) does not allow it. However, the TCP sender can still delay the congestion response by τ allowing the local recovery mechanism to recover from non-congestion event.

5. Local Recovery Mechanisms

The performance benefits to be gained from using the TCP-DCR modifications depend on the existence of an underlying scheme for recovering the losses due to non-congestion events. In case of packet reordering, nothing needs to be done explicitly to recover the reordered packet. In case of wireless networks, we assume that the underlying mechanism is a simple link level retransmission scheme, possibly NACK-based, that does not attempt in-order delivery. Some of the recent research in the area of networking for multimedia [12] also advocate the use of link level retransmission schemes that do not attempt in-order delivery. Alternatively, FEC (Forward Error Correction) schemes could also be used.

6. Steady State Analysis of TCP-DCR

In this section we present an analysis of the steady state bandwidth of TCP-DCR. We use the steady state model with uniform congestion loss probability p.

The congestion window for TCP-DCR can be represented using two functions $f_1(t)$ and $f_2(t)$, where $f_1(t)$ determines the window behavior before the time t_{drop} when a packet is dropped and $f_2(t)$ determines the behavior after the packet drop. The function $f_1(t)$ is the additive increase function. The function $f_2(t)$ has two

components. For the time period τ between t_{drop} and $t_{drop+\tau}$, $f_2(t)$ continues with the additive increase function. Immediately after the congestion delay timer expires, i.e., at $t_{drop+\tau+\epsilon}$, the congestion window is decreased multiplicatively. These two functions can be represented as follows-

$$\begin{aligned} f_1(t) \quad &: w_{t+RTT} \quad \leftarrow \quad w_t + \alpha;\ \alpha > 0 \\ f_2(t) \quad &: w_{t+RTT} \quad \leftarrow \quad w_t + \alpha;\ \alpha > 0, t_{drop} < t < t_{drop} + \tau \\ &\quad w_{t_{drop+\tau}} \quad \leftarrow \quad \gamma * w_{t_{drop+\tau-\epsilon}};\ \gamma > 0, t = t_{drop+\tau} \end{aligned} \tag{2.1}$$

where w_t is the congestion window at time t, RTT is the round trip time, τ is the delay in congestion response and α and γ are constants. Fig. 3 shows the graphical representation of the congestion window against time.

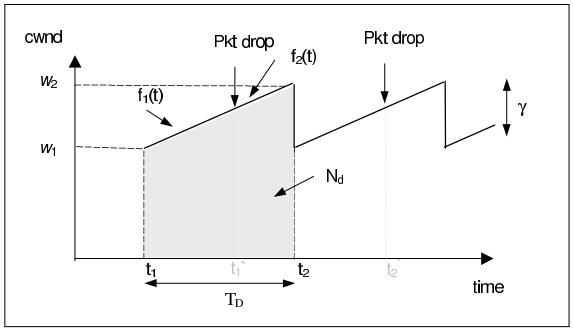


Fig. 3. Analysis of TCP-DCR with No Channel Errors

Let T_D be the time between two successive drops and let N_D be the number of packets sent by the protocol in this time. From equation [2.1], using continuous fluid approximation and linear interpolation of the window between w_t and w_{t+RTT} we get

$$\frac{dw}{dt} = \frac{\alpha}{RTT} \quad \Rightarrow \quad w = \frac{\alpha t}{RTT} + C \tag{2.2}$$

As can be seen from Fig. 3, the parameters T_D and N_D are independent from time shifting the curve along the horizontal (time) axis. This implies that one can arrange it such that a downward interpolation of the curve passes through the origin. That is, without loss of generality and with no change to T_D and N_D , one can set $C = 0$. Thus we have,

$$\begin{aligned} w &= \frac{\alpha t}{RTT} \\ \Rightarrow t &= \frac{wRTT}{\alpha} \end{aligned}$$

The throughput λ (in packets per second) can be given by the number of packets that can be sent between two successive drops (N_D) divided by the time interval between two successive drops (T_D). From the Fig. 3 we have,

$$\begin{aligned} T_D &= t'_2 - t'_1 = \quad t_2 - t_1 \\ &= \frac{RTT}{\alpha}(w_2 - w_1) \end{aligned}$$

The window reduction is determined by the constant γ. Hence we have, $w_1 = \gamma w_2$. Substituting this in the above equation, we get,

$$T_D = \frac{RTT}{\alpha} \cdot w_2 \cdot (1 - \gamma) \tag{2.3}$$

N_D is the shaded are under the curve in Fig. 3. Hence,

$$N_D = \int_{t_1}^{t_2} w(t) \frac{dt}{RTT} = \frac{1}{2\alpha} \cdot w_2^2 \cdot \left(1 - \gamma^2\right) \tag{2.4}$$

However, since N_D is the number of packets between two consecutive drops, the steady state drop probability $p = 1/N_D$.

$$\frac{1}{p} = N_D = \frac{1}{2\alpha} \cdot w_2^2 \cdot \left(1 - \gamma^2\right).$$

$$Thus, w_2 \quad = \quad \sqrt{\frac{2\alpha}{p\,(1-\gamma^2)}} \tag{2.5}$$

Substituting these values in the throughput equation,

$$\lambda \quad = \quad \frac{\sqrt{\frac{\alpha(1+\gamma)}{2(1-\gamma)}}}{RTT\sqrt{p}} \tag{2.6}$$

It is evident from the above result that the throughput of the TCP-DCR protocol is similar to that of TCP Reno [13].

7. Network Dynamics When Congestion Response is Delayed

Even though TCP-DCR responds to a congestion signal with a delay of τ (one RTT) our results indicate that the response to congestion is faster than some of the other proposed protocols [14],[15] which are shown to be TCP-compatible. The earlier studies have shown that even in dynamic network conditions, the slowly responding protocols are fair and safe for deployment [16]. Since DCR responds to congestion faster than these earlier protocols, we expect DCR will be safe even in dynamic network conditions. We demonstrate this point through simulations, later in the chapter.

B. TCP-DCR and Reordering Robustness

In current networks, packet reordering is observed to be non-negligible [3]-[4]. Many new design alternatives for routers or network architectures may benefit if there are no strict restrictions of zero packet reordering. When routers are designed based on parallel forwarding or when multi-path routing is employed, packet reordering could occur. In architectures such as diffserv, requiring no packet reordering restricts the choices for handling the multiple classes of packets of a single flow. As a result, packet reordering is becoming an even more important issue.

1. Related Work

Over the past years, several different measurement studies were conducted to determine the level of packet reordering in the Internet. The measurements were conducted at different network locations using different methodologies during different time periods. These studies have presented observations that are seemingly contradictory to each other.

In [3], the author pioneered the first large-scale measurement study of Internet packets by conducting 20,000 bulk TCP transfers of 100 Kbytes each between 35 Internet sites. In two sets of measurements conducted during December 1994 and November-December 1995, the author found 2% and 0.3% reordering of data packets (0.6% and 0.1% of ACKs) respectively. At least one packet was delivered out of sequence for 36% of the packets in the first measurement and 12% in the second measurement. Other main observations were that reordering was asymmetrical, some paths were sometimes subject to high levels of reordering and the effects were strongly site specific. The two main causes identified for causing the problems were route fluttering and router updates and hence they claimed the reordering behavior was

mainly *pathological* or not very usual.

More recent studies have supported this claim that packet reordering is not a commonly occurring phenomenon on the Internet. An extensive study of packet dynamics for low-bitrate MPEG-4 video streams over paths with more than 5000 routers conducted in November 1999 to May 2000 is presented in [17]. The results of this study indicated that packet reordering is rare. The study presented in [18] looked at 19 million TCP connections on the Sprint backbone and was conducted during February 2002 and October 2002. The results indicated that the packet reordering was observed only in 0.03 to 0.72% of all the data packets (0.15 to 4.9% of all the connections). Additionally, study of packet lag indicated that 87.14% of these reordered would not cause undesirable behavior in TCP connections since the packet lag was less than 3. Finally, measurements made in China [19] during May-June 2003 by tracing 208 connections with 3.3 million data packets using a web-crawler on 10,647 web sites indicated that 5.79% of the sites, and 3.2% of the packets exhibited packet reordering at least once. Of the sites that exhibited reordering 20% of the sites had a reordering frequency of more than 80% indicating strong site dependency. 95.3% if all reordered packets had a lag of less than 3 indicating that the probability of TCP performance degradation was low.

These results were directly contradicted in [20] where the authors claim that packet reordering is not pathological behavior on the Internet and is prevalent at significantly high levels. Their study consisted of measurements conducted on 140 Internet hosts connected to the MAE-EAST exchange during December 1997 and January 1998. The methodology used was significantly different from that in [3], since the authors chose to send back-to-back bursts of 50 ICMP-ping packets of 56-byte for conducting the first measurement and a 100-packet bursts of 512-byte packets for the second measurement. From the first measurement, they observed that the probability

of a session experiencing packet reordering was 90%. From the second measurement they inferred that reordering was a function of network load. Further study indicated that the main cause for reordering was parallelism in Internet components and links due to link-level striping and the multiple paths that a packet can take within the switching devices.

Results from the October 2003 study presented in [21] indicate similar results - that packet reordering was about 56% of all the streams and the leading cause was pointed to be parallelism in the Internet components. Two sets of measurements were conducted, first by sending back-to-back bursts of 50 100-byte UDP packets and second by sending bursts of 100 UDP packets. More reordering was found in the second measurement compared to the first measurement. Another study presented in [4] supported with observations of high levels of packet reordering. The study was conducted using UDP flows in high-speed networks and the authors point to a high correlation between packet rate and observed reordering. They conclude that for high bandwidth applications protocols should be as resilient to packet reordering as they are to packet loss.

Based on these studies, there seem to be two categories of observations, with one set claiming that packet reordering is pathological - it is not prevalent across the Internet, but is rather an artifact of some misconfiguration/misbehavior of network components. Some of these studies observe that packet reordering may be highly local with few sites/links exhibiting high levels of reordering, while the packet reordering is low in case of the general Internet. The other category of results show that packet reordering is not pathological but is widely prevalent and has a possibility of getting worse, since the cause for the reordering is parallelism in Internet components which will only increase as network speed/capacity increases.

The measurement studies that indicate high levels of reordering ([20], [21] and

[4]) used bursts of ICMP or UDP packets for probing while the other measurements were mainly TCP based or used low-bitrate traffic. This indicates that when packets arrive in bursts at a parallel router or switch, it may be characterized by higher packet reordering. This conjecture has been made in [18] as well. Additionally, in [20] authors show that packet reordering is dependent on the network load. This is collaborated in [4] where the authors show that reordering increases as the packet rates increase or conversely, the inter-packet arrival time in the core of the network reduces. This seems to indicate that packet reordering cannot be dismissed easily since the network load on the Internet keeps steadily increasing. The site-dependency of observing higher levels of reordering has been linked to heavy loads - an observation that supports that packet reordering may be a function of network load. As a result, it is important that protocols aimed at high capacity networks be resilient to packet reordering.

Several different solutions have been proposed in literature to solve this problem. In [7] the authors present several schemes which use DSACKs [22] or timestamps [23] are used to identify a *false fast retransmit*, where a packet is falsely identified by the sender as congestion loss and responded with a reduction in sending rate. In response to finding that a false fast retransmit has occurred, the sending rate is restored back to the level it was before the false fast retransmit. The reordering length for the packet is measured using the information available from DSACKs and the *dupthresh* is increased to avoid false fast retransmits. If a RTO timeout occurs, then it is presumed that the *dupthresh* has grown too large and it is reset to 3. In [5] this process is further refined at the cost of maintaining significantly more state at the sender and using complicated algorithms for finding the optimal value for *dupthresh* such that costly RTO timeouts are avoided, while the performance is optimized to provide maximum reordering robustness.

These schemes rely on some additional scheme for identifying reordering in the

network (such as DSACKs or timestamps) and the perceived reordering information is collected from the network to set an optimal value for *dupthresh*. The intent is to estimate the optimal amount of time to delay the triggering of fast retransmit algorithm to provide maximum reordering robustness, without resorting to RTO timeouts too often. By using TCP-DCR, this goal can be met without having to use complex state or algorithms for tuning the value of *dupthresh*. While TCP-DCR does not tune the *dupthresh* based on the perceived reordering in the network, when it is set to one RTT, it provides a simple and effective mechanism for providing reordering robustness without causing RTO timeouts.

2. Simulation Results

We evaluated the performance of TCP-DCR using the ns-2 simulator [24] (version 2.26). The network topology is as shown in Fig. 4. The n different sources are connected to the router R1 which in turn is connected to the router R2. The n different receivers are connected to the router R2. The link between the routers R1 and R2 is configured to be the bottleneck link in experiments simulating congestion in the network. The default values for the link bandwidth and delay for the links between the routers and the end nodes is fixed at 10 Mbps and 1 ms respectively. The link bandwidth and the delay for the bottleneck link is varied in accordance with the requirements of the experiment. Each source i performs bulk data transfer to the receiver i with a packet size of 1000 bytes. DropTail buffer management scheme is used at the routers and the queue size is set to 50 packets, unless otherwise specified.

Packet reordering is simulated by modifying the errormodel object of ns-2 such that randomly selected packets can be delayed for a random amount of time. This allows us the flexibility to choose the percentage of the packets to be delayed, the distributions for choosing the packets randomly as well as the distribution for the

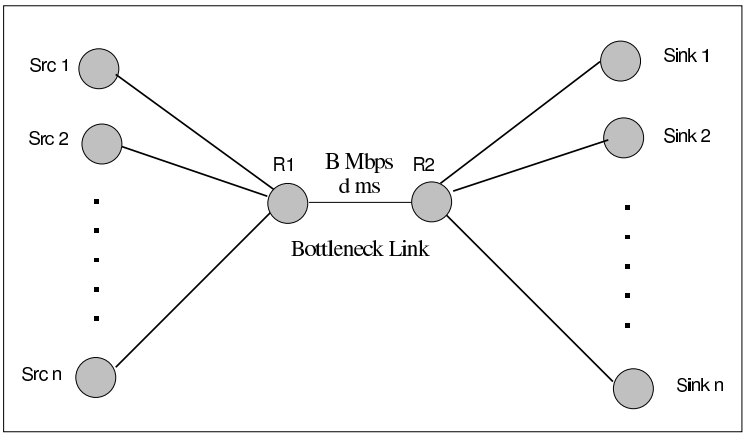


Fig. 4. Network Topology Used in ns-2 Simulations

delays.

The TCP-DCR agent is implemented by modifying the TCP-sack1 implementation of TCP-SACK agent in ns-2. The TCPSink/Sack1 agent is used for the receivers. FTP sources start sending data at time 0 and are staggered to avoid synchronization. All simulations are run for 1100 seconds, but data is collected only after the first 100 seconds to ensure that steady state is reached. The receiver advertises a large window such that the sending rate is not limited by the receiver dynamics.

a. Performance at Varying Packet Delay Rate

One of the primary reasons for reordering in the network is that some of the packets get delayed more than others, and hence arrive out of order. In this experiment we show the effect of delayed packets on TCP-SACK, as the percentage of the packets getting delayed is increased, and the corresponding improvement in performance in case of TCP-DCR. The packet delay is picked from a normal distribution with a mean of 25ms and a standard deviation of 8ms. Thus, most packets chosen for delaying

are delayed in the range 0 to 50ms, simulating mild but persistent reordering. The bottleneck link bandwidth is set to 8Mbps and the delay to 50ms. The receiver advertises a very large window such that the sending rate is not clamped by the receiver dynamics. There is no congestion in the network. The topology consists of a single flow. The experiment is first run with TCP-SACK and repeated for TCP-DCR. The X-axis shows the percentage of packets being delayed, and the Y-axis shows the total throughput in packets for the flow. Fig. 5 shows the results As can be seen

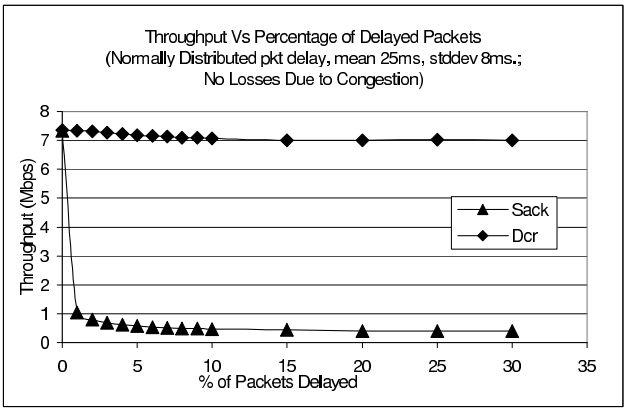


Fig. 5. Throughput vs Percentage of Packets Delayed (with Single Flow)

from the graph, the performance of TCP-SACK degrades rapidly, since the reordered packets are treated as indication of congestion in the network and the sending rate is reduced. Persistent reordering makes the sender congestion window stay at a small value reducing the throughput drastically. TCP-DCR performs significantly better than TCP-SACK. Since there is no congestion in the network, and the packets are only mildly reordered, most packets are recovered within the delay in the congestion response τ. As a results, DCR does not reduce its window and its performance remains close to the performance of TCP-SACK with zero packets delayed in the network.

b. Performance Comparison with Multi-path Routing

One of the situations in which more robustness to reordering is very important is when packets are routed over different paths. Suppose, a router chooses between two different paths for load balancing. In the worst case, alternate packets get routed over the different routes, causing 50% of all packets to get delayed by the difference between the round trip time of the two routes. In this simulation we examine such a situation. The x-axis shows the difference between the RTTs of the two routes. The link delay of the shorter route is fixed at 50ms. Packets are alternately sent over this link and the link with the larger link delay. There is no congestion in the network. Fig. 6 shows the results.

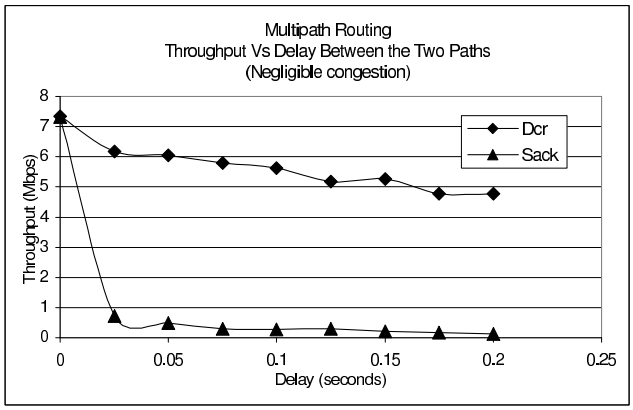


Fig. 6. Performance Comparison with Multi-path Routing

It can be seen from the graph that TCP-DCR performs significantly better than TCP-SACK. When the delay between the two paths becomes larger than the round trip time of the shorter path, the performance of TCP-DCR starts to degrade a little. However, the smoothed RTT estimate at the TCP sender will reflect the average round trip time of the link, and the congestion response delay is scaled by this value. As a result, the performance degradation is not drastic.

c. Performance Comparison with Congestion in the Network

One of the primary concerns with using TCP-DCR is the effect of delaying congestion response on other flows in the network. In order to study this, we conducted several simulations with multiple flows in the network, with half the flows using TCP-DCR and the other half of the flows using TCP-SACK. The graphs are plotted showing the average throughput achieved by each type of flow with an error bar showing the range of throughput for individual flows.

In this experiment, the bottleneck link has a capacity of 10Mbps and a link delay of 10ms. The number of flows in the network is 12, with 6 of them using TCP-DCR and the other 6 using TCP-SACK. Congestion in the network is controlled by varying the buffer size at the router R1 for the link between R1-R2. Fig. 7 shows the results when 10% of the packets are delayed.

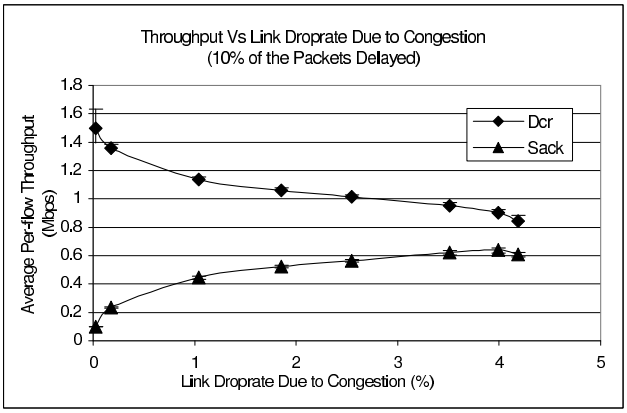


Fig. 7. Throughput vs Link Droprate due to Congestion

Packets are reordered as well as lost due to congestion in this simulation. When a packet is lost due to congestion, the sending rate is reduced both in the case of TCP-SACK and TCP-DCR. However, when a packet is reordered, the sending rate is re-

duced only in the case of TCP-SACK. As a result at low congestion levels, TCP-DCR flows utilize more link capacity than TCP-SACK flows and show better throughput. When the congestion levels in the network increases, the link capacity is more and more equitably shared. It is to be noted that the fact that TCP-DCR realizes better throughput (when packets are reordered) is not due to unfairness, but due to correctly recovering from the reordering events (without reducing the congestion window). We address the fairness issue in Section 1 when we consider zero non-congestion events.

d. Mixed Workload at Varying Packet Delay Rates

We now revisit the experiment showing the performance comparison of TCP-DCR and TCP-SACK at different packet delay rates, but with TCP-DCR and TCP-SACK flows vying for the same bottleneck link capacity. The experimental setup is similar to that explained above, with the only difference that the congestion in the network is fixed at 1.5%. Fig. 8 shows the results.

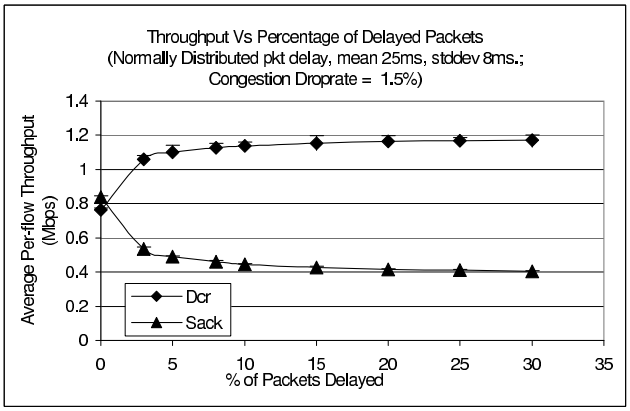


Fig. 8. Mixed Workload at Varying Packet Delay Rates

When there is no reordering in the network, the link capacity is shared equitably by the different flows. When more and more packets are delayed, the TCP-SACK

flows suffer unnecessary reductions in sending rate and their average throughput reduces. The reduction in the average throughput of the TCP-SACK flows is not as drastic as seen in Fig. 5. When one SACK flow reduces its window due to a reordered packet, other SACK flows can claim some of this bandwidth and hence the differences in throughput become smaller with multiple flows. It is observed that TCP-DCR achieves 2-3 times more performance when network reorders packets. It is to be noted again that this gain is not due to DCR's unfairness or aggressiveness, but due to correctly recovering from packet reorder events (when SACK does not do so).

e. Mixed Workload with Varying Number of Flows

In this simulation, we study the throughput of individual flows as the number of flows in the workload is varied. The experimental setup is similar to that mentioned above. Congestion in the network is maintained at around 1% by adjusting the buffer size at router R1 for the link R1-R2, and 1% of the packets are delayed. The number of flows in the network is varied from 4 to 12, with half the flows using TCP-DCR and the other half of the flows using TCP-SACK. Fig. 9 shows the results.

As seen from the results, the difference between the average throughput of TCP-DCR flows and TCP-SACK flows remains fairly stable, indicating that the performance is consistent even when the number of flows in the network is varied.

f. Comparison with Other Protocols

Our results can be directly compared with the results presented in [5] and other solutions proposed for handling reordering. The results show that DCR performs as well or better than the earlier solutions when packet reordering in the network is significant.

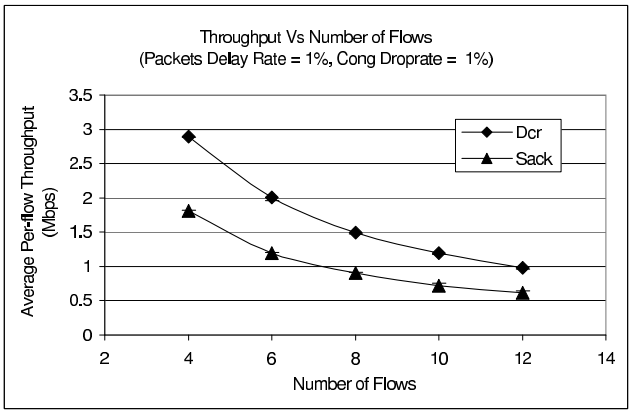


Fig. 9. Mixed Workload with Varying Number of Flows

C. TCP-DCR in a Wireless Network[2]

Wireless networks are characterized by high channel error rates. When TCP is used in wireless networks, the losses due to channel errors are mistaken for congestion losses and the sending rate is unnecessarily reduced, resulting in degraded performance [6]. In this section we show the benefits of using TCP-DCR in wireless networks.

1. Related Work

Several solutions have been proposed to improve the performance of TCP over wireless networks. These solutions fall in one of the following broad categories: (a) Split connection approaches: the connection between the sender and receiver is split into two separate connections, one between the fixed sender and the base station and the other between the base station and the mobile receiver. The losses that are not related to congestion are recovered by the connection between the base station and the mobile host, and hence hidden from the fixed sender. [8],[25],[26],[27] (b) TCP-aware link layer protocols: the link layer is aware of the semantics of the TCP protocol and the dupacks are suppressed from reaching the sender if it can be recovered by link level retransmission. [28],[29] (c) Explicit loss notification approaches : TCP sender relies on the network to provide explicit notification about the error type [30],[31],[32]. (d) Receiver-based approaches : approaches where receivers either delay dupacks [33] or compute the desired sending rate [9]. (e) Modifications to TCP : some of the modified TCP algorithms have been shown to improve the performance of TCP [34],[35]. TCP-DCR modifies the sender side of TCP and relies on link level retransmission for

[2]

recovering from channel errors. Earlier work has shown that local recovery of channel errors is efficient [36].

When both congestion losses and losses due to the transmission errors can occur, the simple solution would be to let the link layer mechanisms to recover from losses due to transmission errors, allowing the transport protocol to recover from congestion losses. In order to maintain the segregation between the different layers of the TCP/IP stack, the link layer should not be required to know the semantics of the transport level protocol and the transport layer should not expect explicit notification about the type of the loss from the network layer. When TCP-DCR is used in wireless networks, a simple link level retransmission scheme that is not aware of TCP semantics would suffice to recover from transmission errors without any explicit notification from the network regarding the type of the loss.

2. Simulation Results

The network topology used in these simulations is similar to the one shown in Fig. 4, except that R2 is the Base station connected to the receivers via wireless links. The default values for the wired link bandwidth and delay is fixed at 100 Mbps and 5 ms respectively. The wireless link bandwidth and delay is kept fixed at 1 Mbps and 20 ms respectively, unless otherwise mentioned.

Link level retransmission is simulated by using the error model and the queue object provided by ns-2. The error model is exponential, and the corrupted packets are buffered at the base station and retransmitted after a delay corresponding to the round trip time of the wireless link, thus simulating link level retransmission. The packet to be retransmitted is added at the head of the queue that holds the packets awaiting transmission. The TCP/IP and MAC layer headers are ignored in the throughput calculations.

a. Performance at Different Channel Error Rates

First, we present the results for the simulation showing the performance improvement offered by TCP-DCR at various channel error rates in Fig. 10. The workload consists of a single flow in this case. There is no congestion in the network.

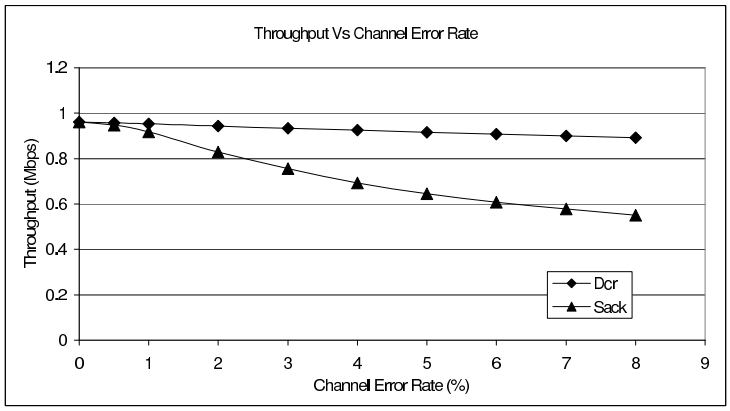


Fig. 10. Throughput vs Channel Error Rate

As can be seen from the graph, TCP-DCR performs significantly better than TCP-SACK. Since there is no congestion in the network, all the packet losses are due to channel errors. Due to delayed response algorithm, TCP-DCR postpones the window reduction upon receiving dupacks. This allows the link layer retransmission scheme time to recover the lost packets thereby making a window reduction unnecessary. Thus, when there is no congestion in the network, the performance of TCP-DCR even at high channel error rates stays close to the performance that can be obtained when there is no channel errors at all. On the other hand, due to repeated reductions of the congestion window, TCP-SACK cannot efficiently utilize the network bandwidth, especially so at high channel error rates.

b. Performance at Different Wireless Delays

Wireless networks have highly varying delays ranging from few milliseconds to few tens of milliseconds for a LAN to several hundred of milliseconds for satellite links[37, 38]. In this section we show the effect of the wireless delay on the performance of the different protocol flavors. The topology is similar to that in the previous section. Fig. 11 shows the results.

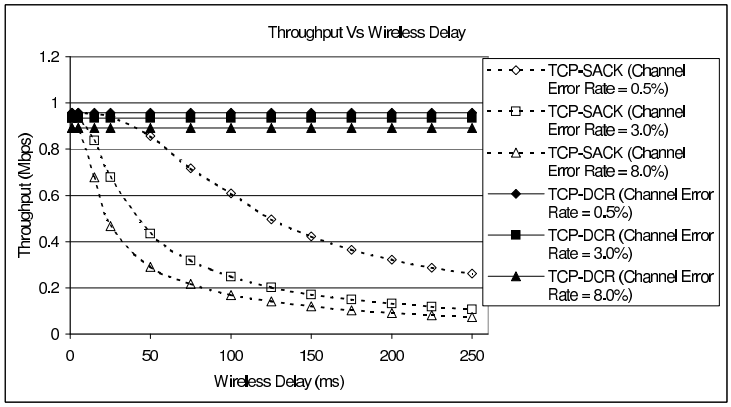


Fig. 11. Throughput vs Wireless Link Delay

It can be seen from the graph that as the wireless link delay is increased, the throughput of the TCP-SACK flows degrades significantly. This is because when the window is reduced incorrectly due to a packet lost by channel errors, it takes a long time for the protocol to increase the window to the correct value again. This results in under-utilization of the network bandwidth. TCP-DCR on the other hand is more robust in the face of large wireless delays, since the window is not reduced as often.

c. Performance Comparison at Different Wireless Bandwidths

Improvement in wireless technology has been constantly raising the bar on how much bandwidth the wireless channels offer. We evaluate the impact of channel bandwidth

on protocol performance. The wireless link delay is fixed at 20ms. The buffer size and the receiver window are adjusted for each simulation to allow maximum link utilization, without causing any congestion. Fig. 12 shows the results.

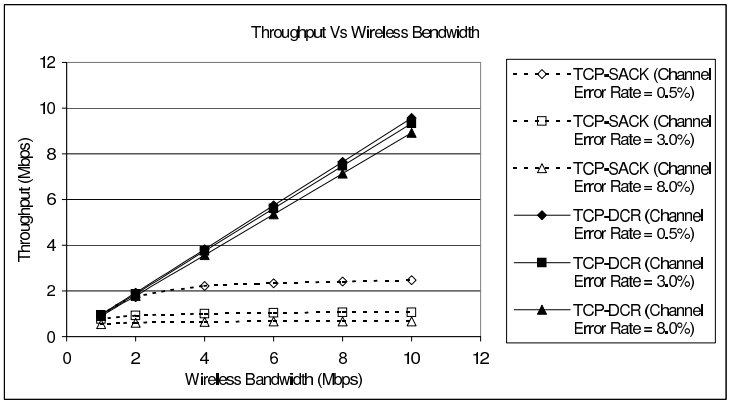


Fig. 12. Throughput vs Wireless Bandwidth

It can be seen from the graph that the TCP-SACK flows cannot utilize the link bandwidth well. At higher channel errors, due to persistent reduction in the sending rate the congestion window remains small, and no matter how much network bandwidth is available, the throughput of the TCP-SACK flows stays almost constant at a small value. TCP-DCR on the the other hand, avoids reducing the congestion window for channel errors and hence, is capable of utilizing the available bandwidth much more efficiently.

d. Performance Comparison with Varying Number of Flows

At this point we take a slight deviation to inspect an important factor to be considered while evaluating the new flavors of TCP protocol - the effect of the number of flows on the simulation results. An important observation made during the above experiments was that TCP flows were not able to completely utilize the bandwidth at high channel

error rates and high wireless delays. It would seem intuitive then that as the number of flows in the network is increased, the utilization of the link could be improved, because when one flow backs off in response to packet loss, some other flow could utilize the link. So we conducted a simulation where the wireless link bandwidth and delay were fixed at 6Mbps and 20 ms, but the number of flows between the source S and the receiver R was increased. The receiver advertised window and the buffer size are adjusted so that a single flow without any losses can almost fully utilize the link. However, note that, when the number of flows is increased, the congestion losses no longer remain zero. The results are presented in Fig. 13.

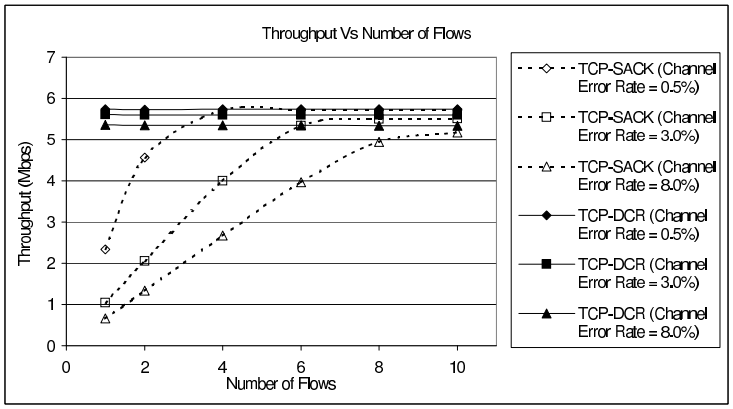


Fig. 13. Throughput vs Number of Flows

As expected, the link utilization does improve at higher number of flows. We have included these results in this paper to demonstrate an important point: results for new protocols shown for just a fixed number of flows are not sufficient. In this case, for the network topology that we have chosen, by having a fixed number of flows greater than 8, TCP-SACK could be shown to provide very good performance even at very high channel error rates.

Another perspective on this issue can be provided by the following argument.

It has been shown in [13] that the throughput of the TCP protocol is proportional to $\frac{1}{RTT*\sqrt{p}}$ (when timeouts are ignored), where p is the loss rate seen by a TCP flow and RTT is the round trip time perceived by the TCP sender. When there is no congestion in the network, p represents only channel errors for TCP-SACK . These losses do not depend on the number of flows in the network, and are fixed relative to the number of flows in the network. Then for any particular value of p and RTT, the throughput obtained by a TCP source is fixed, say at T. The fair share of bandwidth for any particular flow when there are n different flows in the network is B/n. When the value of n is chosen such that $B/n \leq T$, it will appear as if the protocol is making the best utilization of the available bandwidth, irrespective of how the protocol treats the channel errors.

Consider TCP-DCR on the other hand. As shown in the Eq. 2.6, the throughout of a TCP-DCR flow is also proportional to $\frac{1}{RTT*\sqrt{p}}$. However, in this case, p primarily represents the loss rate due to congestion in the network. As a result, when congestion in the network is zero, the throughout is only controlled by the receiver's window. In other words, when there is no congestion in the network, the TCP-DCR can effectively utilize all the available bandwidth, irrespective of the number of flows in the network.

It might be tempting at this point to suggest that all we need, to improve the performance of TCP on a wireless network, is to fill up the pipe with many flows such that all the bandwidth can be utilized. This could probably be a feasible solution if we can ensure that at all the times there will be enough flows in the network to keep it fully utilized. However, if that is not the case, and we wish to have maximum utilization irrespective of how many flows are in the network, then we would require modifications to existing TCP protocols. Also, wireless technology is improving at a rapid rate, and as new technology becomes available, the bandwidth keeps increasing. The higher bandwidth would require larger number of flows to keep the link fully

utilized for the same channel error rate. It would be unreasonable to depend only on the number of flows in the network to make the best use of the available bandwidth.

e. Performance with Congestion in the Network

In this set of simulations, the workload consists of 24 flows, half of which use TCP-DCR and the other half use TCP-SACK. The different levels of congestion are obtained by varying the buffer size at the router R1. The bottleneck link capacity is set to 10Mbps and the delay to 5ms. The wireless link bandwidth and delay ate 1Mbps and 20ms. Fig. 14 shows the results. In the graph, congestion loss rates of less than 1% are labeled as low error, in the range of 2.5-3.5% are labeled as moderate congestion and greater than 3.5% are labeled as high congestion.

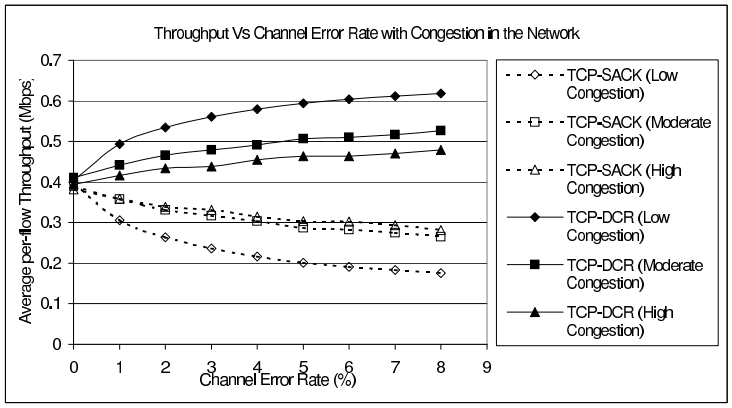


Fig. 14. Throughput vs Channel Error Rate with Congestion in the Network

It can be seen from the figure that when the congestion loss rate is low, the average throughput of the TCP-DCR flows is far more than that of TCP-SACK flows. This is not because the TCP-DCR flows are more aggressive than TCP-SACK. Rather, it is due to the fact that the TCP-DCR flows can make use of the link bandwidth *not utilized effectively* by the TCP-SACK flows. Recall from the discussion

in previous sections that the TCP-SACK flows cannot utilize the available bandwidth completely at high channel errors because of persistent window reductions. The TCP-DCR flows claim this share of the bandwidth not used by the TCP-SACK flows. So when the congestion in the network is low, the TCP-DCR flows help improve the link utilization without starving the TCP-SACK flows.

The throughput achieved by TCP-DCR flows is inversely proportional to the congestion loss rate in the network, whereas the throughput of the TCP-SACK flows is inversely proportional to the sum of the congestion loss rate and the channel error rate. So, as the congestion loss rate in the network increases, the difference in the average throughput of the TCP-DCR flows in the network compared to that of the TCP-SACK flows becomes narrower.

f. Performance Comparison on Satellite Links

Satellite links are characterized by very high wireless delays, with the one way delays being as large as 250ms [37]. With such high delays, when the congestion window is reduced unnecessarily in response to channel errors, it takes a long time to recover the window back to the optimal size. Thus the performance of TCP-SACK degrades drastically in satellite networks as the channel error increases. In this section we present the results of the simulations for performance comparison on satellite links. The network topology is similar to that above, except that the wireless link has a large one way delay of 250ms, making the end-to-end RTT 520ms. The average link drop rate due to congestion is in the range of 0.1-0.4%. Fig. 15 shows the results, demonstrating the performance improvements with TCP-DCR.

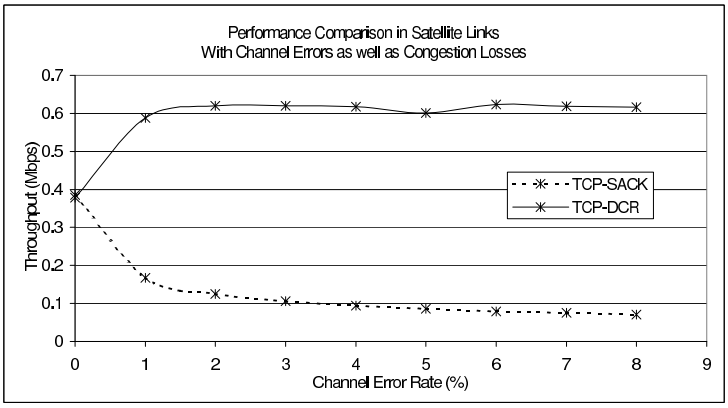


Fig. 15. Performance Comparison over Satellite Links

g. Comparison of TCP-DCR with Other Protocols

We have carried out extensive simulations to compare the performance of DCR with other protocols, particularly, TCP-Reno and TCP-Westwood. Due to lack of space, we have included only one of the results, showing the performance comparison of TCP-DCR with TCP-Westwood at different wireless delays and channel error rates in Fig. 16. The WestwoodNR agent was used in this simulation in the ns-2.26 version. The wireless link bandwidth is fixed at 1Mbps and the receiver advertised window and the wireless link buffer size are adjusted to maximize the link utilization even at large delays. Congestion losses occur only at the bottleneck link router. The simulations indicate that at low channel errors and low delays, the performance of both the protocols flavors are similar. At higher channel error rates and large delays, TCP-DCR performs better.

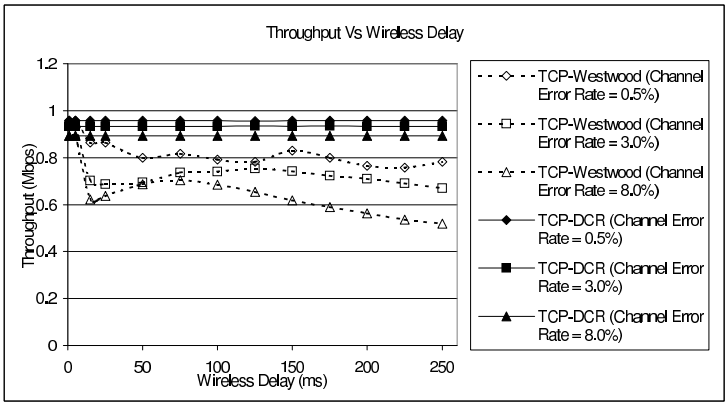


Fig. 16. Performance Comparison of TCP-DCR vs TCP-Westwood

D. TCP-DCR in Networks with Only Congestion Events

TCP-DCR delays the response to congestion by one RTT. The delay is chosen to be a tradeoff between providing robustness to non-congestion events while retaining responsiveness to congestion events. In order to understand the issues related to deployment, it is important to study the behavior of TCP-DCR in networks with no non-congestion events and only congestion events. In this section we present the results of such a study.

1. Simulation Results

The results we present here are from simulations where the losses are only due to congestion. We study the behavior of TCP-DCR at three different levels - (a) flow level - throughput (relative fairness) when TCP-SACK and TCP-DCR flows compete with each other, time taken to relieve and reclaim bandwidth for sudden changes in available bandwidth and interaction with web-like transfers. (b) protocol level - Packet Delivery time and RTT estimation for individual flows. (c) the network level - average queue lengths and drop rates at the bottleneck link.

The topology used for these experiments is as shown in Fig. 17. The links between the sources and the router are high-capacity wired links with bandwidth 100Mbps, delay 5ms and buffer size equal to the delay-bandwidth-product. The link between the router and the base station is the wired bottleneck link of capacity 10Mbps and delay 5ms. The links between the base station and the receivers are wireless with capacity 1Mbps, delay 20ms and queue-length of 50 packets. Congestion level on the bottleneck link is modified by varying the buffer size on the link between the router and the base station. The receiver advertised window is set such that in the absence of congestion at the bottleneck link, the per-flow throughput does not

exceed the wireless link capacity to ensure that the congestion happens only on the link between the router and the base station. Each source performs a single bulk data transfer to the corresponding receiver with a packet size of 1000 bytes. The duration for the ftp transfer for most experiments in this paper is set to 1100 seconds, but for the experiments inspecting the behavior at the flow level and the queue level, the transfer duration is smaller - 200 seconds - due to the large amount of data being collected. The total number of flows in the network is 24 (unless otherwise mentioned).

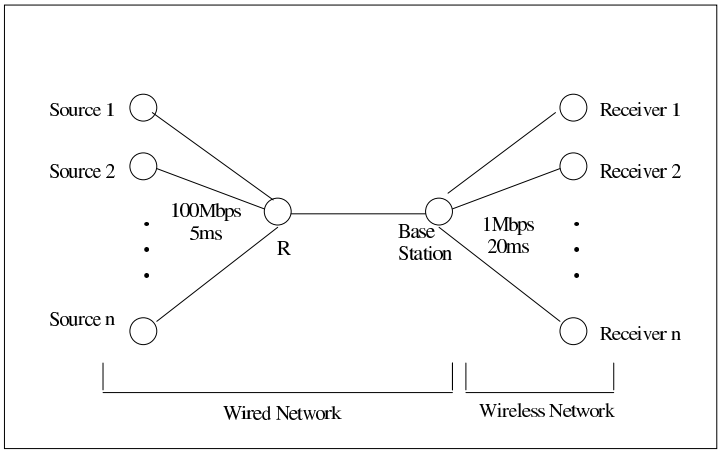


Fig. 17. Network Topology for Experiments with Congestion Losses

a. Performance Comparison at Different Congestion Loss Rates

In this experiment we evaluate the interaction between 12 TCP-DCR and 12 TCP-SACK flows. Fig. 18 shows the average throughput of the TCP-DCR flows in comparison with the average throughput of the TCP-SACK flows.

As can be seen from the graph, the TCP-DCR flows share the bottleneck link with the TCP-SACK flows in a relatively fair manner. For long-term flows, delaying the

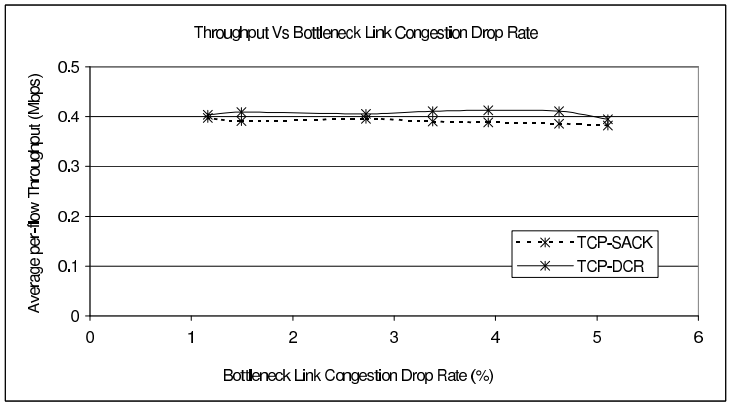


Fig. 18. Throughput vs Congestion Loss Rate

congestion response by one RTT does not make TCP-DCR more aggressive compared to the TCP-SACK flows. TCP-DCR is observed to respond to congestion faster than some of the other proposed protocols [14],[15] which are shown to be TCP-compatible. The earlier studies have shown that even in dynamic network conditions, the slowly responding protocols are fair and safe for deployment [16]. Since TCP-DCR responds to congestion faster than these earlier protocols, we expect TCP-DCR will be safe even in dynamic network conditions.

b. Performance Comparison for Sudden Changes in Available Bandwidth

In this experiment we evaluate the performance of TCP-DCR in comparison with TCP-SACK for sudden changes in the available bottleneck bandwidth. The network consists of 24 flows. Half the flows do long-term ftp transfer starting at time 0 seconds using the protocol being evaluated. The other half of the flows carry shorter ftp transfer (referred henceforth as traffic) using TCP-SACK starting at 50 seconds and lasting for 50 seconds. Thus, 50 seconds after the long-term flows are started, the available network bandwidth goes down by 50%. At 100 seconds, the traffic stops,

and the available bandwidth doubles back to the original level. The average link drop rate over the period of the simulation is about 2%. Fig. 19 shows the aggregate throughput of the long term flows and the traffic (computed with 1 second bins) against time. From the figure it is clear that the response of TCP-DCR to sudden fluctuations in traffic is similar to that of TCP-SACK.

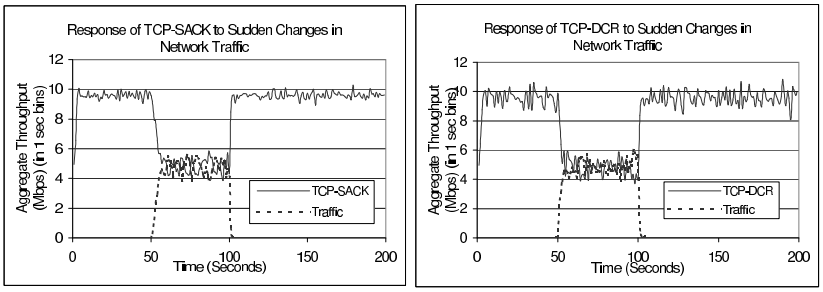


Fig. 19. Throughput vs Time for Sudden Changes in Traffic

In order to quantify the reaction time to sudden changes in load, we computed the time it takes for existing flows to drop down to 55% of the link capacity, thus allowing the new flows to achieve 45% of the link capacity. The time to reach (55%, 45%) allocation for TCP-SACK was 5.89 seconds and for TCP-DCR, it was 3.80 seconds. This shows that TCP-DCR is not worse than TCP-SACK in responding to sudden increases in traffic load.

c. Interaction with Web-like Traffic

In this section we evaluate the performance of TCP-DCR and TCP-SACK when competing with a traffic mix of several short-term flows simulating web-transfers. The network consists of 8 long-term ftp flows(TCP-SACK or TCP-DCR) and 500 web-like flows(TCP-SACK). The transfers are started at around 0 seconds with a staggering of 1ms to avoid synchronization. Each short-term flow sends N packets

after T seconds from the start of its previous transfer. N is drawn from a uniform distribution between 10 and 20 and T is drawn from a pareto distribution with mean 15 seconds, simulating the different request sizes and user think-times. The random variable generators for the short-term flows are seeded with the flow id, so that any given flow has a fixed pseudo random sequence. This ensures that when the simulation is first run with TCP-SACK ftp transfers and then repeated with TCP-DCR ftp transfers, the random variables used in simulating the web transfers, have the same value. The average link drop rate over the period of the simulation is 3%. Fig. 20 shows the aggregate throughput of the long term flows and the traffic (computed with 1 second bins) against time.

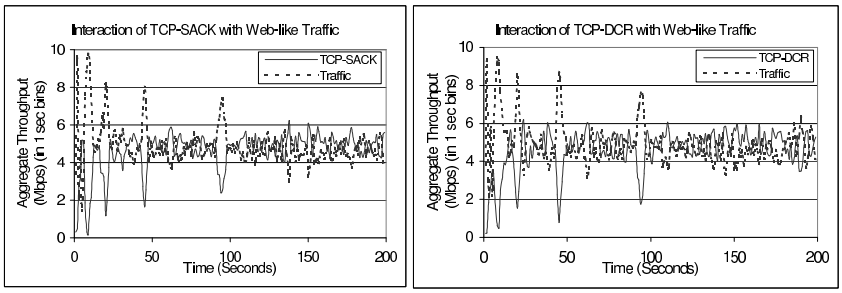


Fig. 20. Interaction with Web-like Traffic

In the case of TCP-SACK, the aggregate throughput of TCP-SACK flows over the simulation period is 4.76Mbps, and that for the web traffic is 4.84Mbps. The aggregate throughput of TCP-DCR flows is 4.73Mbps and that for the web traffic is 4.82Mbps. This indicates that the interaction of the TCP-DCR flows with short-term web traffic is similar to that of TCP-SACK.

d. Packet Delivery Time

In this section and the next we take a look at some of the protocol level dynamics. Since the TCP-DCR protocol delays the triggering of the congestion recovery algorithms by one RTT, it is possible that the packet delivery time during congestion is increased by up to one RTT. When there is no congestion in the network, the packet delivery time is unaffected. In this section we present the results of simulations verifying the packet delivery time for the TCP-DCR flows in comparison to the TCP-SACK flows. Three separate simulations are considered - in the first, all 24 flows are TCP-SACK, in the second all 24 flows are TCP-DCR and in the third, half the flows (i.e.,, 12 flows) are TCP-SACK and the other half are TCP-DCR. This allows us to compare the packet delivery time for TCP-DCR with that of TCP-SACK, and also examine the effect of TCP-DCR flows on the packet delivery time of TCP-SACK flows when the workflows consists of a mix of the two flavors. The average congestion drop rate at the bottleneck link is maintained at about 3.3% by using a buffer size of 70 packets at the bottleneck link. Fig. 21 shows the plot of packet delivery times for a randomly chosen TCP-DCR/TCP-SACK flow against the packet sequence number.

The plots show that the packet delivery times are scattered in two regions. The dense population of points around 60-100ms represent the packets that are delivered normally. The points with larger delay represents packets delayed due to larger instantaneous queue lengths and the packets that are recovered through retransmission. In the first simulation where all the flows are TCP-DCR the average packet delivery time for packets of the sample flow recovered via retransmission is 398ms. In the second simulation where all the flows are TCP-SACK flows, it is 302ms. In the third simulation where 50% flows are TCP-DCR and the other 50% are TCP-SACK, the

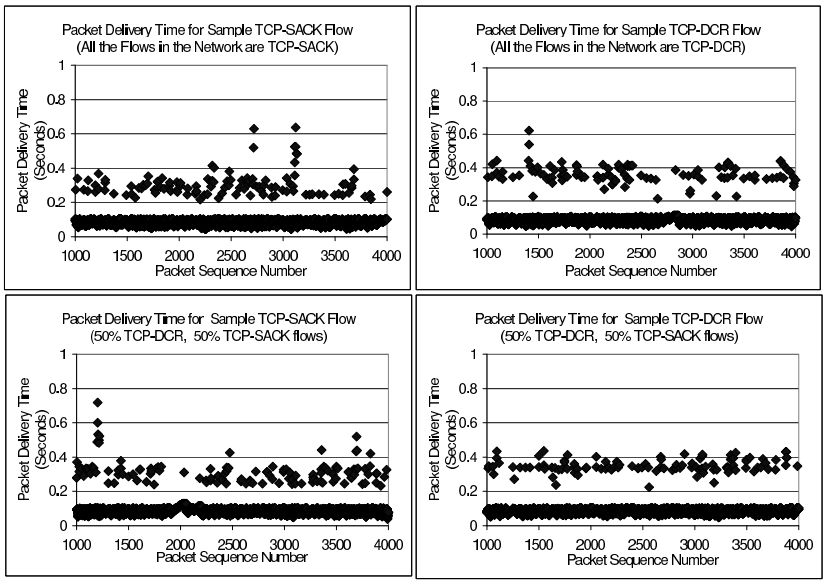


Fig. 21. Packet Delivery Time

average packet delivery time for retransmitted packets of the sample TCP-DCR flow is 356ms and for TCP-SACK, it is 296ms. We notice from these observations that the recovery time for a retransmitted packet in case of the sample TCP-DCR flow is about one RTT more than that of the sample TCP-SACK flow. Also, we notice that when the workload consists of a mix of TCP-DCR and TCP-SACK flows, the time to recover a packet through retransmissions for TCP-SACK is not affected, compared to the simulation with all TCP-SACK flows.

e. RTT Estimates

As explained in the above section, delaying the congestion response of TCP by one RTT can increase the packet recovery time of lost packets. The packet delivery time for the rest of the packets is similar to that in any standard implementation of TCP. According to Karn's algorithm used by most standard implementations of TCP, a

retransmitted packet is not used in estimating the round trip time. Thus the delayed congestion response of TCP-DCR does not affect the RTT estimation of TCP. Fig. 22 shows the plot of instantaneous RTT, smoothed RTT and RTT variance for a randomly chosen TCP-DCR/TCP-SACK flow against the packet sequence number. The results agree with the discussion presented here.

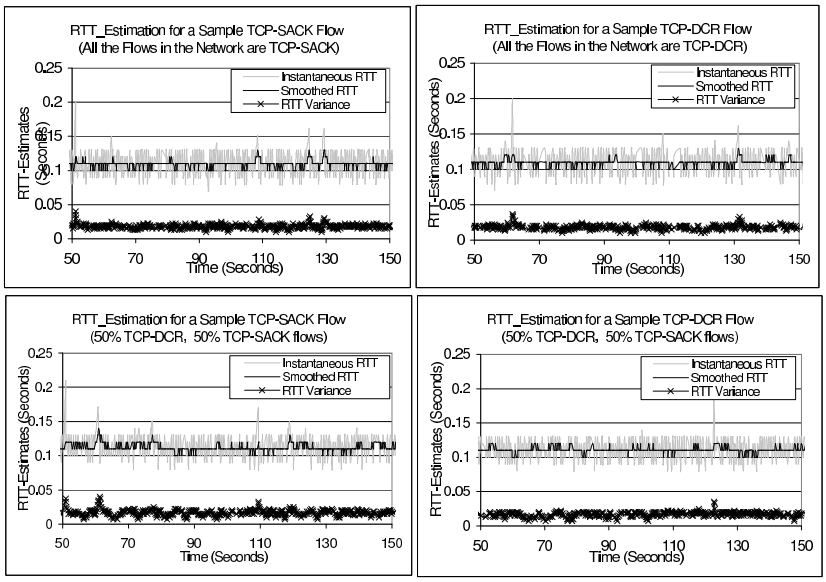


Fig. 22. RTT Estimation

f. Effect on Network Queue Lengths

In this section, we evaluate the effect of TCP-DCR flows on the bottleneck link queue length. The network topology is similar to that in the above section. The average bottleneck link drop rate is about 3.3 - 3.4%. Fig. 23 shows the plot of the instantaneous and the average queue length at the bottleneck link.

With 24 flows in the network, the DropTail queue at the bottleneck link is almost full all the time irrespective of whether the flows are TCP-DCR or TCP-SACK. Thus

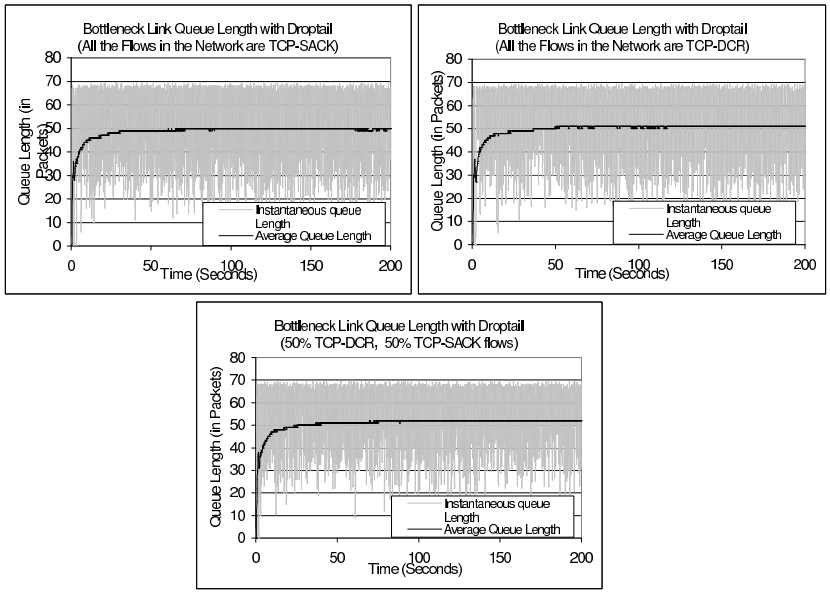


Fig. 23. Bottleneck Link Queue Length with DropTail Queue Management

it is hard to evaluate the impact of TCP-DCR on the queue lengths. The average queue length varies slightly (51 packets when all flows are TCP-DCR, 50 packets when all the flows are TCP-SACK and 52 packets for the mixed workload), but the difference is negligible.

To further investigate this matter, we replaced the queue management scheme at the bottleneck link router with RED. The *minthresh_* and *maxthresh_* parameters are set to 25% and 75% of the total buffer size. Fig. 24 shows the plot of the instantaneous and the average queue length at the bottleneck link.

It can be seen from this graph, that the queue length does not change much. The average queue lengths are 36, 34 and 35 packets, when all flows are TCP-DCR, TCP-SACK or a mixture of the two respectively.

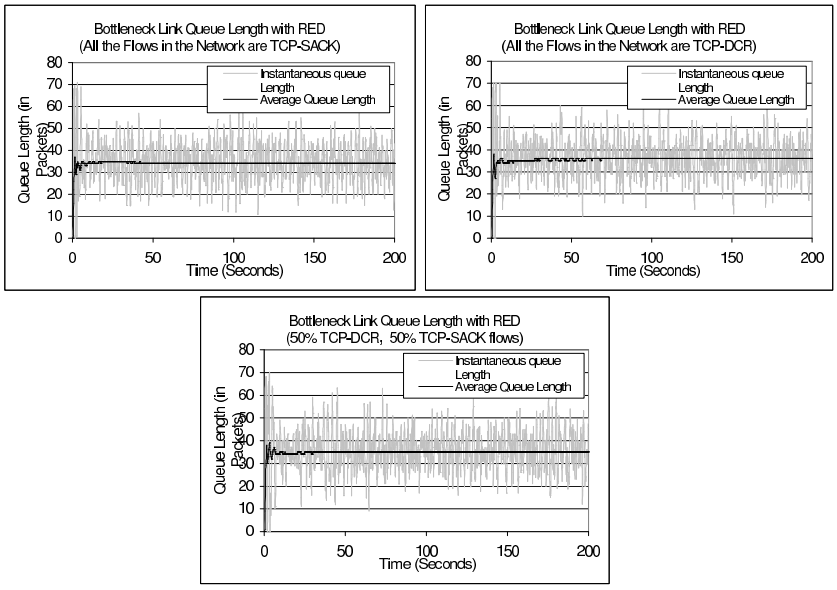


Fig. 24. Bottleneck Link Queue Length with RED Queue Management

g. Effect on Bottleneck Link Congestion Loss Rate

One of the primary concerns when protocol characteristics are modified is the effect the modifications have on the network. TCP-DCR delays the response to loss notification. Hence, it is interesting to study how an increase in the offered load effects the congestion drop rate on the bottleneck link. For this simulation, we keep all the other parameters constant and vary the number of flows in the network and study the congestion drop rate at the bottleneck link. Note that the receiver window is adjusted such that the per-flow throughput is always less than the capacity of the wireless link and hence the congestion occurs only at the bottleneck link. The buffer size at the bottleneck link between the router and the base station is fixed at 50 packets to ensure that a wide range of congestion drop rates may be observed, as the number of flows is varied. The simulations were conducted across the three traffic workloads considered in the earlier sections. The first graph in Fig. 25 shows the

results. TCP-SACK (100,0), TCP-DCR (0,100) represent the average link drop rate when all the flows in the network are TCP-SACK and TCP-DCR respectively. TCP-SACK (50,50) and TCP-DCR (50,50) represent the average drop rates observed by TCP-SACK flows and TCP-DCR flows respectively when the workload consists of a mix of both the flows. It can be seen from the graph that the average congestion loss rate observed for TCP-DCR is similar to that of TCP-SACK.

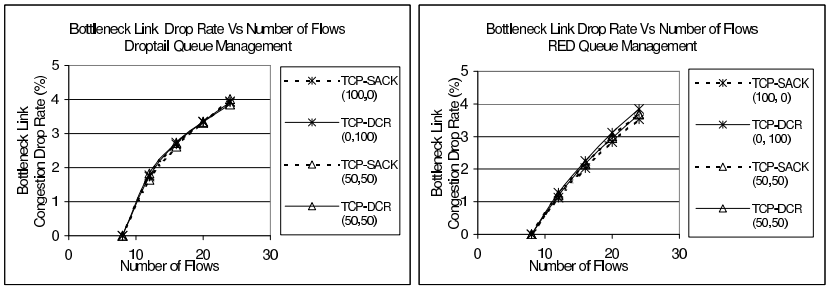


Fig. 25. Bottleneck Link Congestion Loss Rate vs Number of Flows

Again, in the interest of being comprehensive, we repeated this experiment with RED queue management scheme at the bottleneck link. The second graph in Fig. 25 shows the results. In the previous section we noticed that the average queue length is slightly different in the three cases. In an RED queue, the drop probability depends on the average queue length and hence the average drop probability varies slightly for the three cases, but the difference is fairly negligible.

E. Conclusions

In this chapter, we proposed TCP-DCR that employs delayed congestion response and local recovery to recover from non-congestion events. We have provided an analysis to show that TCP-DCR achieves similar throughput as regular TCP in steady state conditions. We studied TCP-DCR's handling of non-congestion events in two specific

cases, namely, packet reordering and wireless channel errors. In both cases, results indicate that TCP-DCR offers significantly better performance by simply delaying congestion response for one RTT. We then studied the impact of employing TCP-DCR in networks with zero non-congestion events. Our evaluation at multiple levels - individual flows, TCP characteristics and network characteristics - has shown that TCP-DCR does not significantly impact other flows or the network even when all the packet losses are due to congestion alone. Based on these results, TCP-DCR seems to offer a simple, unified solution to handle non-congestion events safely.

CHAPTER III

TCP ON HIGH BDP LINKS WITH LOW LEVELS OF MULTIPLEXING

Over the past few decades, the traffic on the Internet has increased by several orders of magnitude. However, the Internet still remains a stable medium for communication. This stability has been attributed primarily to the wide-spread use of congestion control algorithms of TCP [39]. The congestion control algorithms are designed such that bandwidth is shared fairly among flows with similar RTTs. However, these same congestion control algorithms hold TCP back from scaling to future networks with high-bandwidth links of the order of several Gbps.

The TCP congestion control algorithms use *additive increase multiplicative decrease*(AIMD) for moderating the congestion window. When there are no losses in the network, the window is increased by one for each RTT. Upon a loss of packet, the window is reduced by half. In other words, the AIMD parameter for increase is chosen such that the available bandwidth is probed conservatively, but the parameter for decrease upon a packet loss, is chosen to give up the bandwidth aggressively. While this has worked well for the networks in the past, for future networks that have capacity of the order of several hundreds of Mbps or Gbps and are shared by few flows, this could lead to highly degraded performance.

To illustrate the problem effectively, consider the following popular example - The throughput of a TCP connection is given by $T \simeq \frac{1.2*S}{R\sqrt{p}}$, where S is the packet size, R is the round trip time for the connection and p is the packet loss rate [13]. This means that for a standard TCP connection using a packet size of 1500 bytes over a connection with round trip delay of 200ms and packet loss rate of $1.0X10^{-5}$, the maximum throughput that can be achieved is 23.2Mbps. If the packet loss rate were reduced to $1.0X10^{-7}$ the maximum throughput could be increased to 232.4Mbps.

Conversely, to achieve a throughput of 1Gbps, the packet loss rate required should be $5.4X10^{-9}$ or lower and for 10Gbps it should be $5.4X10^{-11}$. These loss rates are unreasonable. For a 10Gbps link, the loss rate translates to a loss of at most one packet in $1.85X10^{10}$ packets or at most one loss for every six hours ! Clearly, the standard TCP connections do not scale in high capacity networks.

In this chapter, we present a simple layering technique for the existing congestion response algorithms to make it scale in high-bandwidth networks. The idea of layering to probe and utilize the available bandwidth has been researched earlier in the context of video transmission on the Internet and in multicasting[40, 41]. The contribution of this paper is to extend this idea to the congestion control algorithms in TCP so that scalability can be achieved at the cost of minimal implementation overhead, while retaining many of the desirable characteristics that have made TCP the protocol of choice.

A. LTCP : Using Layered Congestion Control for Improving TCP Performance in High BDP Networks with Low Multiplexing

The conservative AIMD algorithm used in the traditional flavors of TCP have worked remarkably well in the the past, because very high capacity links (greater than several hundreds of Mbps or Gbps) were available only at the core of the network where several thousands of flows got multiplexed. But in the recent past, there has been an increase in the use of high capacity links for connecting research labs end-to-end for fast exchange of large amounts of data. The availability of inexpensive gigabit NICs and fast computers indicate that an average end user could have access to high capacity networks, in the not-too-distant future. In such an environment, where the density of flows is low and the available per-flow capacity is high, the TCP mechanism

of increasing by just one packet per RTT tends to be too conservative, while at the same time the window reduction by a factor of half tends to be too drastic. This results in inefficient link utilization.

1. Related Work

Over the past few years, several solutions have been put forth for solving the problem of performance degradation on high-speed networks due to the use of TCP. These solution can be classified into four main categories - a) Tuning the network stack b) Opening parallel TCP connections between the end hosts c) Modifications to the TCP congestion control and d) Modifications to the network infrastructure or use of non-TCP transport protocol.

The traditional solution to improve the performance of TCP on high-capacity networks has been to tune some of the TCP parameters. Several auto-tuning schemes have been proposed such as [42], [43], [44] and [45]. [46] presents a comparison of some of these auto-tuning schemes. Tuning the stack improves the performance of TCP in high-speed networks significantly and could be used in conjunction with other schemes to achieve the best possible performance.

A number of other proposals have employed *network striping*, where the *application* is network-aware and tries to optimize the network performance by opening parallel TCP connections. Some of the applications that use this approach or allow it as an option are XFTP [47], GridFTP [48], storage resource broker [49] and [50]. In [51] the authors provide a library called *PSockets* (Parallel Sockets) to make it easier to develop applications that use network striping. While most of this work has been at the application level, in the MulTCP scheme[52] the authors present a mechanism where a single TCP flow behaves as a collection of several virtual flows. In [53], the authors describe a scheme for using virtual round trip time for choosing a tradeoff

between fairness and the effectiveness of network usage. In [54] the authors describe a scheme for managing the striped TCP connections that could take different network paths. However, all the above mentioned schemes use a fixed number of parallel connections and choosing the *optimal* number of flows to maximize the performance without effecting the fairness properties is a significant challenge.

The third category of research for improving the performance of TCP in high-speed networks has been to modify the congestion response function of TCP itself. High-Speed TCP [55] uses a congestion window response function that has a higher slope than TCP. Scalable TCP [56], uses multiplicative increase/multiplicative decrease response, to ensure that the congestion window can be doubled in a fixed number of RTTs. FAST TCP [57] relies on the delay-based bandwidth estimation of TCP Vegas [58] and is optimized for Gbps links. Bic-TCP [59] focuses on the RTT fairness properties by modifying the congestion response function using binary search with additive increase and multiplicative decrease. H-TCP [60] aims to identify whether the congestion window is operating in the low speed mode or high-speed mode and uses two different values for the increase/decrease parameters accordingly.

Several other schemes that go beyond modifications only to TCP have also been proposed. In the XCP[61] scheme, the authors propose changes to the TCP congestion response function as well as the network infrastructure. In schemes like Tsunami[62], RBUDP[63] and SABUL/UDT [64] reliable data transfer is achieved by using UDP for data and TCP for control information. GTP[65] is also a rate based protocol, but focuses on max-min fair rate allocation across multiple flows to support multipoint-to-point data movement.

In this chapter, we propose the Layered TCP scheme which modifies the congestion response function of TCP at the sender-side and requires no additional support from the network infrastructure or the receivers. In a sense, this scheme can be

thought of as an emulation of multiple flows at the transport level, with the key contribution *that the number of virtual flows adapt to the dynamic network conditions*. Layering schemes for probing the available bandwidth have been studied earlier in the context of multicast and video transfer, for example [40, 41]. LTCP, in contrast to this earlier body of work, uses layering *within the congestion control algorithm of TCP with per-ACK window adaptation* to provide efficient bandwidth probing in high bandwidth links while retaining fairness between multiple flows with similar RTTS.

2. Layered TCP: The Framework

LTCP is based on the very simple concept of *virtual layers* or *virtual flows*. To start out with, every LTCP flow has only one layer. If the sending rate of the flow increases, without observing any losses, then based on some criteria, it increases the number of layers and continues to do so until a loss event is observed. When operating at any given layer K, the flow behaves as if it were a collection of K virtual flows, increasing the aggressiveness of probing for bandwidth. Just like the standard implementations of TCP, the LTCP protocol is ACK-clocked and the congestion window of an LTCP flow changes with each incoming ACK. However, since the LTCP flow operating at layer K emulates K virtual flows, it increases the congestion window by K packets per RTT. This is similar to the increase behavior explored in [52].

For determining the number of layers that a flow should operate at, the following scheme is used. Suppose, each layer K is associated with a step-size δ_K. When the current congestion window exceeds the window corresponding to the last addition of a layer (W_K) by the step-size δ_K, a new layer is added. Thus,

$$W_1 = 0$$

$$W_2 = W_1 + \delta_1$$

...

$$W_K = W_{K-1} + \delta_{K-1} \tag{3.1}$$

and the number of layers = K, when $W_K \leq W < W_{K+1}$. Fig. 26 shows this graphically.

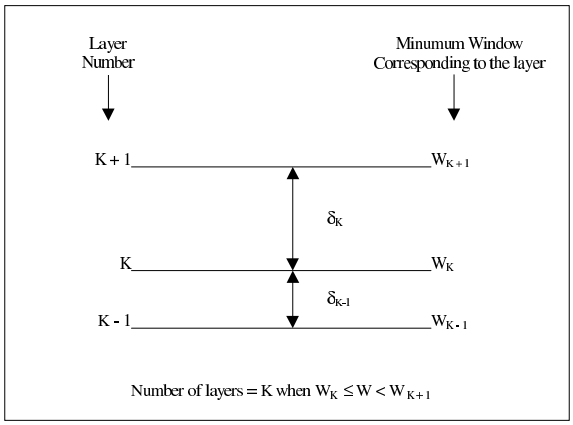


Fig. 26. Graphical Perspective of Layers in LTCP

The step size δ_K associated with the layer K should be chosen such that convergence is possible when several flows share the bandwidth. Consider the simple case when the link is to be shared by two LTCP flows. Say, the flow that started earlier operates at a higher layer K_1 (with a larger window) compared to the later-starting flow operating at a smaller layer K_2 (with the smaller window). In the absence of network congestion, the first flow increases the congestion window by K_1 packets per RTT, whereas the second flow increases by K_2 packets per RTT. In order to ensure that the first flow does not continue to increase at a rate faster than the second flow, it is essential that the first flow adds layers at a rate slower than the second flow. Thus, if δ_{K_1} is the step-size associated with layer K_1 and δ_{K_2} is the step-size associated with layer K_2, then

$$\frac{\delta_{K_1}}{K_1} > \frac{\delta_{K_2}}{K_2} \tag{3.2}$$

when $K_1 > K_2$, for all values of $K_1, K_2 \geq 2$.

The design of the decrease behavior is guided by a rather similar reasoning - in order for two flows starting at different times to converge, the time taken by the larger flow to regain the bandwidth it gave up after a congestion event should be larger than the time it takes the smaller flow to regain the bandwidth it gave up. Suppose the two flows are operating at layers K_1 and K_2 ($K_1 > K_2$), and ω_1 and ω_2 is the window reduction of each flow upon a packet loss. After the packet drop, suppose the flows operate at layers $K_1^{'}$ and $K_2^{'}$ respectively. Then, the flows take $\frac{\omega_1}{K_1^{'}}$ and $\frac{\omega_2}{K_2^{'}}$ RTTs respectively to regain the lost bandwidth. From the above reasoning, this gives us -

$$\frac{\omega_1}{K_1^{'}} > \frac{\omega_2}{K_2^{'}} \tag{3.3}$$

The window reduction can be chosen proportional to the current window size or be based on the layer at which the flow operates. If the latter is chosen, then care must be taken to ensure convergence when two flows operate at the same layer but at different window sizes.

Eq. 3.2 and 3.3 provide a simple framework for the congestion response function of TCP for the congestion avoidance phase in high-speed networks. The congestion window response in slow start is not modified, allowing the protocol to evolve with experimental slowstart algorithms such as [66]. At the end of slowstart the number of layers to operate at can easily be determined based on the window size.

Asymptotic convergence to fairness among flows of similar RTT can be assured for competing flows if they satisfy the constraints in Eq. 3.2 and 3.3. It must be noted

however, that when the flows have different RTTs, the above scheme could make the RTT unfairness worse than that of TCP. Since LTCP is ACK-clocked similar to TCP, the window of an LTCP flow with shorter RTT grows faster than that of an LTCP flow with larger RTT. In addition, since each layer K is associated with a step size δ_K, it takes exactly δ_K/K RTTs for a flow to increase the number of layers to $(K+1)$. When two flows with different RTTs share a bottleneck link, the flow with the short RTT will be able to add layers faster than the flow with the larger RTT making the RTT unfairness worse. In order to compensate for this dependence of aggressiveness on RTT, we introduce the RTT compensation factor K_R and modify the per-ACK behavior such that, an LTCP flow at layer K will increase the congestion window at the rate of $K_R * K$ for the successful receipt of one window of acknowledgments. The RTT compensation factor is made proportional to the RTT. In order to ensure that the flows do not become aggressive as queues build up, we make K_R dependent only on the propagation delay of the link. In our experiments we use the lowest measured RTT sample for choosing the value of K_R. This uniformly scales up the rate at which a flow increase its window, with respect to RTT. Since the RTT compensation factor K_R is constant for a given RTT and multiplicative in nature, it does not alter the Eq. 3.2 and 3.3 for the flows operating at same RTT.

In order to develop a protocol that can use the above framework, the key is to determine the appropriate relationships for the step size δ (or equivalently, the window size W_k at which the layer transitions occur) and the window reduction that satisfy the conditions in Eq. 3.2 and 3.3. Several different design choices are possible. We evaluate in this work one particular design that lets us retain the AIMD nature of TCP.

3. Design Choice

In order to evaluate the effectiveness of the LTCP protocol, we present the following design option. We support this design with extensive analysis to understand the protocol behavior. However, this is by no means the only possible or the best possible design for the LTCP scheme. The scheme presented here is a means for illustrating the effectiveness of the LTCP framework for efficiently claiming available bandwidth, without sacrificing the convergence and fairness properties.

First we determine appropriate values for the step size δ (or equivalently, the window size W_k at which the layer transitions occur) and the window reduction that satisfy the conditions in Eq. 3.2 and 3.3. Based on this choice, the RTT compensation factor K_R can be determined to provide RTT fairness to satisfy a given requirement (e.g.. similar to TCP or similar to rate based flow etc.).

For determining the parameters, we start with the decrease behavior and analyze behavior for flows with similar RTTs. Since, the key requirement we have is to retain the AIMD properties of TCP, the decrease behavior is chosen to be multiplicative. The window reduction is based on a factor of β such that -

$$\omega = \beta * W \tag{3.4}$$

Based on this choice for the decrease behavior we determine the appropriate increase behavior such that the conditions in Eq. 3.2 and 3.3 are satisfied. To provide an intuition for choice of the increase behavior, consider Eq. 3.3

$$\frac{\omega_1}{K_1'} > \frac{\omega_2}{K_2'} \tag{3.5}$$

In order to allow smooth layer transitions, we stipulate that after a window reduction due to a packet loss, at most one layer can be dropped i.e., a flow operating

at layer K before the packet loss should operate at either layer K or $(K-1)$ after the window reduction. Based on this stipulation, there are four possible cases -

1. $K_1^{'} = K_1, K_2^{'} = K_2$
2. $K_1^{'} = (K_1 - 1), K_2^{'} = K_2$
3. $K_1^{'} = K_1, K_2^{'} = (K_2 - 1)$ and
4. $K_1^{'} = (K_1 - 1), K_2^{'} = (K_2 - 1)$.

It is most difficult to maintain the convergence properties, when the larger flow does not reduce a layer but the smaller flow does, i.e.,, $K_1^{'} = K_1, K_2^{'} = (K_2 - 1)$.

With this worst case situation, Eq. 3.3 can be written as -

$$\frac{\omega_1}{K_1} > \frac{\omega_2}{(K_2 - 1)} \tag{3.6}$$

If this inequality is maintained for adjacent layers, we can show by simple extension, that it can be maintained for all other layers. So consider $K_1 = K, K_2 = (K-1)$. Then, the above inequality is

$$\frac{\omega_1}{K} > \frac{\omega_2}{K - 2} \tag{3.7}$$

Suppose, the window for flow 1 is $W^{'}$ when the packet loss occurs and the window of flow 2 is $W^{''}$ then, substituting Eq. 3.4 in the above equation, we have,

$$\begin{aligned} \frac{W^{'}}{K} &> \frac{W^{''}}{K-2} \\ \Rightarrow W^{'} &> \frac{K}{K-2} W^{''} \end{aligned} \tag{3.8}$$

In order for the worst case behavior $(K_1^{'} = K, K_2^{'} = (K-2))$ to occur, the window $W^{'}$ could be close to the transition to the layer $(K+1)$ and the window $W^{''}$ could have recently transitioned into layer $(K-1)$. In order to get the estimate of

the worst case we substitute these values in the above equation to get -

$$W_{K+1} > \frac{K}{K-2} W_{K-1} \tag{3.9}$$

Based on this, we conservatively choose the increase behavior to be

$$W_K = \frac{K+1}{K-2} W_{K-1} \tag{3.10}$$

Note that alternate choices are possible. This is essentially a tradeoff between efficiently utilizing the bandwidth and ensuring convergence between multiple flows sharing the same link. While it is essential to choose the relationship between W_K and W_{K-1} such that the condition in Eq. 3.34 is satisfied to ensure convergence, a very conservative choice would make the protocol slow in increasing the layers and hence less efficient in utilizing the bandwidth.

Now suppose we choose to add the second layer at threshold $W_2 = W_T$. Then, by recursively substituting, we have

$$W_K = \frac{K(K+1)(K-1)}{6} W_T \tag{3.11}$$

By definition, $\delta_K = W_{K+1} - W_K$ and hence we have,

$$\delta_K = \frac{K(K+1)}{2} W_T \tag{3.12}$$

By simple substitution, we can show that the inequality in Eq. 3.2 is satisfied. Also, since this scheme was designed with the worst case for the inequality in Eq. 3.3, that condition is satisfied as well, when two competing flows are at adjacent layers. The result for adjacent layers can then be easily extrapolated for non-adjacent layers. It can also be shown that when two flows operate at the same layer, the inequality in Eq. 3.3 is satisfied.

a. Choice of W_T and β

The choice of the threshold W_T, the window size at which the LTCP flows starts to increase the number of layers, determines the region where the increase of LTCP has similar increase behavior as TCP. We choose a value of 50 packets for W_T. This value is motivated by the fact that when the window scale option [67] is not turned on, the maximum window size allowed is 64Kb which is about 44 packets (of size 1500 bytes). The window scale option is used in high-speed networks, to allow the receiver to advertise large window size. In slower networks, when the window scale option is not turned on, the actual sending rate is capped by this window value. We choose to begin the aggressive bandwidth probing of LTCP beyond this threshold.

The relationship between W_K and K has been derived based on the stipulation that after a window reduction due to packet drop, at most one layer is dropped. In order to ensure this, we have to choose the parameter β carefully. The worst case for this situation occurs when the flow has just added the layer K and the window $W = W_K + \Delta$, when the packet drop occurs. In order to ensure that the flow does not go from layer K to $(K-2)$ after the packet drop, we need to ensure that

$$\beta W_K < \delta_{K-1} \tag{3.13}$$

(Ignoring the reduction due to Δ since we are computing the worst case behavior.) On simple substitution, this yields,

$$\beta < \frac{3}{K+1} \tag{3.14}$$

Thus, β should be chosen such that the above equation is satisfied. The first two columns in Table I shows the number of layers corresponding to the window size at

layer transitions (W_K) with $W_T = 50$. For a 2.4Gbps link with an RTT of 150ms and packet size of 1500 bytes, the window size can grow to 30,000. The number of layers required to maintain full link utilization is therefore $K = 15$. Based on this, we conservatively choose $\beta = 0.15$ (corresponding to $K = 19$).

With this design choice, LTCP retains AIMD behavior. At each layer K, LTCP increases the window additively by K, and when a packet drop occurs, the congestion window is reduced multiplicatively by a factor of β.

b. Time to Claim Bandwidth and Packet Recovery Time

The primary goal for designing the LTCP protocol is to be able to utilize available link bandwidth aggressively in high-speed networks. Here, we provide quantitative analysis for time taken by an LTCP flow (in terms of RTTs) for claiming available bandwidth and the packet loss recovery time. For this analysis, we consider the case where the RTT compensation factor $K_R = 1$.

Suppose the maximum window size corresponding to the available throughput is W_K. Then, time to increase the window to W_K can be obtained as the sum of the time to transition from layer 1 to 2, 2 to 3 and so on until layer K. In other words, the time to increase the window to W_K is -

$$T(\delta_1) + T(\delta_2) + ... + T(\delta_{K-2}) + T(\delta_{K-1})$$

where $T(\delta_K)$ is the time (in RTTs) for increasing the window from layer K to $(K+1)$. When the flow operates at layer K, to reach to the next layer, it has to increase the window by δ_K and the rate of increase is K per RTT. Thus $T(\delta_K)$ is given by $\frac{\delta_K}{K}$. Substituting this in the above equation and doing the summation we find that the

time to reach a window size of W_K is

$$T(\delta_1) + \frac{(K-2)(K+3)}{4} W_T \tag{3.15}$$

Note that the above analysis assumes that slowstart is terminated before layering starts. The third column in Table I shows the speedup in claiming bandwidth compared to TCP, for an LTCP flow with $W_T = 50$, with the assumption that slowstart is terminated when window $= W_T$.

Table I. Comparison of LTCP (with $W_T = 50$ and $\beta = 0.15$) to TCP

K	W_K	Speedup in Claiming Bandwidth	Speedup in Packet Loss Recovery Time
1	0	-	-
2	50	1.00	1.00
3	200	2.00	6.67
4	500	2.57	10.00
5	1000	3.17	13.33
6	1750	3.78	16.67
7	2800	4.40	20.00
8	4200	5.03	23.33
9	6000	5.67	26.67
10	8250	6.31	30.00
11	11000	6.95	33.33
12	14300	7.60	36.67
13	18200	8.25	40.00
14	22750	8.90	43.33
15	28000	9.56	46.67
16	34000	10.21	50.00
17	40800	10.87	53.33
18	48450	11.52	56.67
19	57000	12.18	60.00
20	66500	12.84	63.33

An LTCP flow with window size W will reduce the congestion window by βW. It then starts to increase the congestion window at the rate of at least $(K-1)$ packets per RTT (since we stipulate that a packet drop results in the reduction of at most one layer). The packet loss recovery time then, for LTCP is $\frac{\beta W}{(K-1)}$. In case of TCP, upon a packet drop, the window is reduced by half, and after the drop the rate of increase is 1 per RTT. Thus, the packet recovery time is $W/2$. The last column of Table I

shows the speed up in packet recovery time for LTCP with $\beta = 0.15$ compared to TCP. Based on the conservative estimate that a layer reduction occurs after a packet drop, the speed up in the packet recovery time of LTCP compared to TCP is a factor of $3.33 * (K - 1)$.

c. Response Function

In order to understand the relationship between throughput of an LTCP flow and the drop probability p of the link, and provide the basis for determining the value of K_R, we present the following analysis. Fig. 27 shows the steady state behavior of the congestion window of an LTCP flow with a uniform loss probability model. Suppose the number of layers at steady state is K and the link drop probability is p. Let W'' and W' represent the congestion window just before and just after a packet drop respectively. On a packet loss the congestion window is reduced by $\beta W''$. Suppose the flow operates at layer K' after the packet drop. Then, for each RTT after the loss, the congestion window is increased at the rate of K' until the window reaches the value W'', when the next packet drop occurs. Since we stipulate that at most one layer can be dropped after the window reduction due to packet loss, the window behavior of the LTCP flow, in general, will look like Fig. 27 at steady state.

With this model, the time between two successive losses, say T_D, will be $\frac{\beta W''}{K_R K'}$ RTTs or $\frac{\beta W''}{K_R K'} * RTT$ seconds. The number of packets sent between two successive losses, say N_D, is given by the area of the shaded region in Fig. 27. This can be shown to be -

$$N_D \simeq \frac{\beta (W'')^2}{K_R K'} (1 - \frac{\beta}{2}) \tag{3.16}$$

The throughput of such an LTCP flow can be computed as $\frac{N_D}{T_D}$. That is,

$$BW = \frac{W''}{RTT} (1 - \frac{\beta}{2}) \tag{3.17}$$

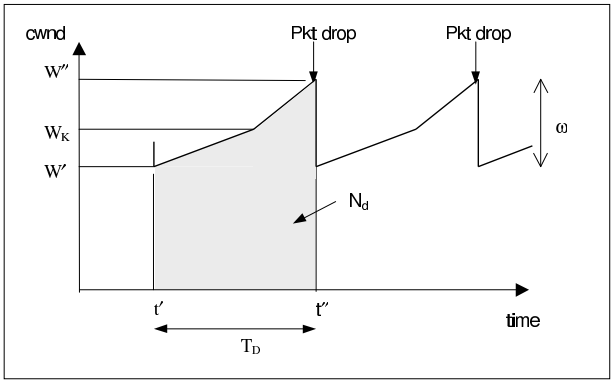


Fig. 27. Analysis of Steady State Behavior

The expected number of packets sent between two losses N_D is $\frac{1}{p}$. By substituting this in Eq. 3.16, and solving for $W^{''}$ we have,

$$W^{''} = \sqrt{\frac{K_R K^{'}}{\beta(1-\frac{\beta}{2})p}} \tag{3.18}$$

$K^{'}$ is a discrete integer value, based on the Eq. 3.11. Hence it can be approximated by $\lfloor(\frac{6W^{''}}{W_T})^{\frac{1}{3}}\rfloor$. Substituting this in Eq. 3.18, we have,

$$W^{''} = (\frac{K_R(\frac{6}{W_T})^{\frac{1}{3}}}{\beta(1-\frac{\beta}{2})p})^{\frac{3}{5}} \tag{3.19}$$

Substituting in Eq. 3.17 we have

$$BW = \frac{C.K_R^{\frac{3}{5}}}{RTT.p^{\frac{3}{5}}} \tag{3.20}$$

$$where \quad C = \frac{(\frac{6}{W_T})^{\frac{1}{5}}(1-\frac{\beta}{2})}{[\beta(1-\beta)]^{\frac{3}{5}}}$$

Fig. 28 shows the response function of LTCP when K_R is equal to 1. Response function of other high-speed proposals as well as that of unmodified TCP are shown for comparison. [60] states that the response function of H-TCP is similar to that of

HS-TCP. From the figure, it can been seen that the slope of the response function of only LTCP is similar to that of regular TCP indicating that LTCP behaves in the AIMD fashion similar to TCP. At the same time, the curve is shifted along the Y-axis, indicating better scalability in high-speed networks, compared to TCP.

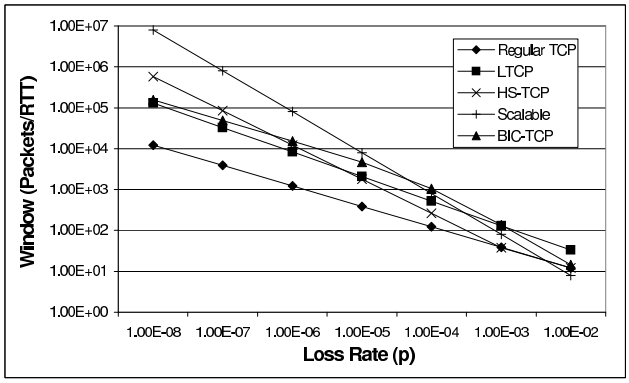


Fig. 28. Response Function of Different High-speed Protocols

d. RTT Unfairness and Choice of K_R

In this section we assess the RTT unfairness of LTCP under the assumptions of random loss model as well as synchronized loss models. Based on the discussion for the synchronized loss model, we derive the relationship between K_R and RTT to achieve RTT unfairness similar to that of TCP.

In [68], for a random loss model the probability of the packet loss λ is shown to be -

$$\lambda \propto \frac{A(w, RTT)}{A(w, RTT) + B(w, RTT)} \tag{3.21}$$

where $A(w, RTT)$ and $B(w, RTT)$ are the window increase and decrease functions respectively.

For LTCP $A(w, RTT) = K/w$ and $B(w, RTT) = \beta W$, when RTT compensation is not used. Substituting these values in the above equation and approximating $K \propto W^{1/3}$, we can calculate the loss rate λ as -

$$\lambda \propto \frac{1}{(1 + \beta W_s^{5/3})} \tag{3.22}$$

where W_s is the statistical equilibrium window.

It is clear from the above equation that the two LTCP flows experiencing the same loss probability, will have the same equilibrium window size, regardless of the round trip time. However, throughput at the equilibrium point becomes inversely proportional to its round trip time since the average transmission rate r_s and is given by W_s/RTT.

The loss probability for TCP with similar assumptions is given by:

$$\lambda \propto \frac{1}{(1 + 0.5W_s)} \tag{3.23}$$

The equilibrium window size of the TCP flow does not depend on the RTT either. Therefore, for random losses the RTT dependence of window of an LTCP flow is same as TCP. Thus LTCP has window-oriented fairness similar to TCP and will not perform worse than TCP in case of random losses.

Because of the nature of current deployment of high bandwidth networks, it is likely that the degree of multiplexing will be small and as such, an assumption of synchronized loss model may be more appropriate. So we present here the details of the analysis with the synchronized loss model as well.

Following a similar analysis in [59], for synchronized losses, suppose the time between two drops is t. For a flow i with round trip time RTT_i and probability of

loss p_i, the average window size is

$$W_i = \frac{\frac{1}{p_i}}{\frac{t}{RTT_i}} = \frac{RTT_i}{tp_i} \tag{3.24}$$

since the flow will send $\frac{1}{p_i}$ packets between two consecutive drop events and the number of RTTs between the two consecutive loss events $\frac{t}{RTT_i}$.

From Eq. 3.20, we have the bandwidth of an LTCP flow to be

$$\begin{aligned} BW = \frac{W_i}{RTT_i} &= \frac{C.K_{R_i}^{\frac{3}{5}}}{RTT_i.p^{\frac{3}{5}}} \\ \Rightarrow p_i &= \frac{C^{\frac{5}{3}}K_{R_i}}{W_i^{\frac{5}{3}}} \end{aligned} \tag{3.25}$$

where C is a constant.

By substituting the above in Eq. 3.24 and simplifying we get,

$$W_i = (\frac{tK_{R_i}}{RTT_i})^{\frac{3}{2}}.C^{\frac{5}{2}} \tag{3.26}$$

When the RTT unfairness is defined as the throughput ratio of two flows in terms of their RTT ratios, the RTT unfairness for LTCP is -

$$\frac{(\frac{W_1}{RTT_1})(1-p_1)}{(\frac{W_2}{RTT_2})(1-p_2)} \simeq \frac{\frac{W_1}{RTT_1}}{\frac{W_2}{RTT_2}} = (\frac{RTT_2}{RTT_1})^{\frac{5}{2}}(\frac{K_{R_1}}{K_{R_2}})^{\frac{3}{2}} \tag{3.27}$$

(since $p << 1$).

The above equation shows that by choosing K_R appropriately, the RTT unfairness of LTCP flows can be controlled. For instance, choosing $K_R \propto RTT^{\frac{1}{3}}$, the RTT unfairness of the LTCP protocol will be similar to the AIMD scheme used in TCP. By choosing $K_R \propto RTT$, the effect of RTT on the scheme can be entirely eliminated and the LTCP protocol behaves like a rate controlled scheme independent of the RTT. By choosing an intermediate value such as, $K_R \propto RTT^{\frac{1}{2}}$, we can reduce the RTT unfairness of LTCP in comparison to TCP.

In general, suppose we choose, K_R proportional to RTT^{α}, where α is a constant. After a window reduction ω, suppose a flow operates at layer $K^{'}$. When RTT compensation is used it takes $\frac{\omega}{K_R * K^{'}}$ RTTs or $\frac{\omega * RTT}{K_R * K^{'}}$secs to regain the lost bandwidth. Suppose two flows with different RTTs are competing for the available bandwidth, Eq. 3.3 can be re-written as

$$\frac{\omega_1 * RTT_1}{K_{R1} * K_1^{'}} > \frac{\omega_2 * RTT_2}{K_{R2} * K_2^{'}}$$

Substituting the value of ω and further solving it, we have,

$$\Rightarrow (\frac{RTT_1}{RTT_2})^{\alpha-1} < \frac{(K^{'}+1)}{K^{'}}$$
$$\Rightarrow (\alpha - 1) < log(1 + \frac{1}{K^{'}}) \quad \Rightarrow \alpha <= 1 \tag{3.28}$$

The above equation has been derived by assuming a worst case RTT ratio of 10 while taking the logarithm. It shows that when the RTT compensation factor is chosen based on the relationship $K_R \propto RTT^{\alpha}$ the value of α should be less that or equal to 1, to ensure asymptotic convergence. Since we aim to keep the behavior of LTCP similar to that of TCP we choose $K_R \propto RTT^{\frac{1}{3}}$.

e. Router Buffer Requirements

It has been conventional wisdom to set the router buffer size based on the classical rule of thumb of delay-bandwidth product of the link. For links with high bandwidth and high delays, this choice makes the required router buffer size very large. Research on sizing the router buffers [2] have shown that the classical rule of thumb is based on the desire to maintain high link utilization on the link, when a single flow tries to saturate the link. Since LTCP uses a less drastic decrease rule compared to TCP,

the buffer size required by a single LTCP flow for keeping the link fully utilized all the time, is lower than that of TCP. Based on analysis similar to that in [2] it can be shown that the buffer size requirements for a single LTCP flow is $\frac{\beta}{(1-\beta)}(C * 2T_p)$ where C is the link capacity and T_p is the propagation delay of the link. Since we choose the value of β to be 0.15, the minimum buffer size required is $0.176 * C * 2T_p$, an 82% reduction compared to that of TCP.

f. Implementation Details

The LTCP protocol requires simple sender-side changes to the congestion window response function of TCP. For the chosen design, it uses two additional parameters - W_T and β. In our implementation, W_T and β are set to 50 and 0.15 respectively. The window thresholds W_K, computed using the parameter W_T, may be saved in an array for quick lookup. When a new connection is established, the protocol is started with $K = 1$ and the slowstart algorithm of standard TCP. When slowstart is exited, the number of layers K is obtained based on the current cwnd by looking up the table containing the values of W_K.

Both the framework and the design use discrete values for layers. However, in the implementation, K can be made continuously variable by linearly increasing it over δ_K. The pseudocode for the LTCP algorithm is given below.

```
if(srtt < min_rtt)
{
      min_rtt = srtt
      recompute   K_R
}
if(newack)
{
      K_fract = (cwnd − W_K)/delta_K
      cwnd = cwnd + (K_R * (K + K_fract))/cwnd
      if(cwnd >= W_K+1)
            K++
}
```

$if(packet \quad loss)$
$\{$
$\quad cwnd = cwnd(1 - \beta)$
$\quad if(cwnd < W_K)$
$\quad\quad K - -$
$\}$

The rest of the algorithms used in the traditional implementation of TCP - for instance the algorithms for RTT calculations, SACK processing, timer management etc, are not modified.

4. NS-2 Simulation Results

To evaluate the LTCP protocol, we conducted experiments based on both simulations on the ns-2 simulator and emulations on a Linux test-bed. The ns-2 simulations help us evaluate the behavior of LTCP under diverse network conditions. The emulations, on the other hand, help us evaluate a real implementation of LTCP on the Linux network stack. Third party evaluation of LTCP in comparison with other advanced TCP stacks such as High-Speed TCP [55], Scalable TCP [56], FAST TCP [57], Bic-TCP [59] and H-TCP [60] and UDP based scheme UDT [64] is currently underway at the SLAC lab at Stanford. In this section we focus on the simulation results.

Fig. 29 shows the network topology used in the simulations. The topology is a simple dumbbell network. The bottleneck link bandwidth is set to 1 Gbps unless otherwise specified. The links that connect the senders and the receivers to the router have a bandwidth of 2.4Gbps. The end-to-end RTT is set to 120ms. The routers have the default queue-size of 5000 packets unless specified otherwise. DropTail queue management is used at the routers. The LTCP protocol is implemented by modifying the TCP/Sack1 agent. The unmodified TCP/Sack1 agent is used for TCP. The receiver advertised window is set to a large value to ensure that it does not interfere with the simulations. For the LTCP flows, the parameter W_T is set to 50 packets and

the parameter β was set to 0.15. The traffic constitute of FTP transfer between the senders and receivers.

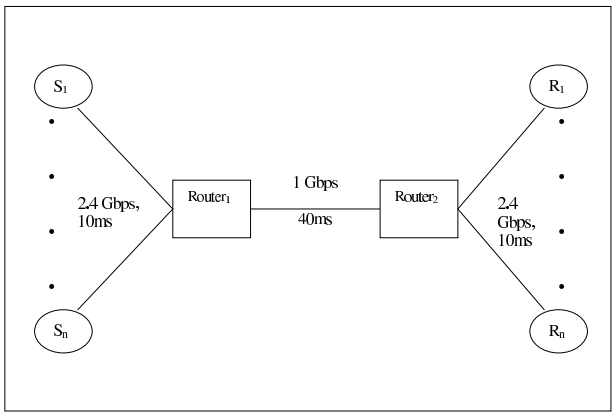
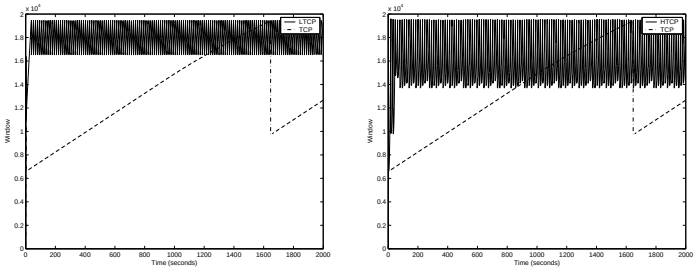


Fig. 29. Simulation Topology

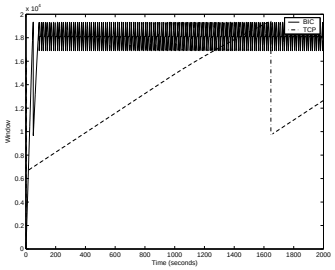
a. Basic Comparison with TCP

Since LTCP uses adaptive layering, it is capable of increasing its window size to the optimal value much faster than TCP. Also, when a packet loss occurs, the window reduction of LTCP is not as drastic as TCP. As a result the window adaptation of LTCP is much more efficient in utilizing the link bandwidth in high-speed networks. Fig. 30 shows congestion window of LTCP in comparison with that of TCP, when the network consists of only one flow. As seen from the figure, the congestion window of LTCP reaches the optimal value several orders of magnitude faster than the TCP flow. The comparison of windows for HTCP and BIC with TCP are included for reference. Since HTCP chooses the window reduction dynamically in the range (0.5, 0.8), the overall fluctuation in the HTCP window is slightly worse than that of LTCP and BIC.



(a) LTCP

(b) HTCP



(c) BIC

Fig. 30. Comparison of Congestion Window

Table II shows the comparison between the goodput and average packet loss rates for different protocols at different bottleneck link bandwidths. The throughput is calculated over a period of 2000 seconds after the flow reaches steady state. Due to the large period for averaging the throughput and the buffer size of 5000 packets, TCP flow seems to be able to obtain reasonably high throughput. As seen from Fig. 30, due to the large width of the TCP window cycle, using a smaller duration for measuring throughput would yield lower TCP throughput, depending on the window size during the measurement period. Since the high speed protocols operate close to the optimal value most of the time, the link utilization will be high even if the measurement duration is reduced. However, due to operating close to the optimal value, the congestion loss rate observed by high-speed flows is larger than that of TCP which due to under-utilization of the link sees lower congestion losses. Among the different high-speed protocols, LTCP and BIC have lower self-induced losses compared to HTCP while achieving similar goodput.

Table II. Goodput and Packet Loss Rate for Different High-speed Protocols

Bottleneck Link Bandwidth	TCP		LTCP		HTCP		BIC	
	Goodput (Mbps)	Packet Loss Rate (%)	Goodput (Mbps)	Packet Loss Rate (%)	Goodput (Mbps)	Packet Loss Rate (%)	Goodput (Mbps)	Packet Loss Rate (%)
100Mb	89.12	7.51E-05	96.15	4.07E-02	96.12	3.92E-02	95.24	6.39E-02
250Mb	222.91	8.89E-06	240.38	1.21E-02	240.37	1.77E-02	238.10	1.65E-02
500Mb	437.01	3.01E-06	480.77	4.83E-03	478.28	1.06E-02	476.19	5.22E-03
1Gb	817.58	7.47E-07	961.54	1.95E-03	952.10	6.32E-03	952.38	1.50E-03
2.4Gb	2103.07	1.18E-07	2307.69	6.10E-04	2277.29	3.15E-03	2277.45	2.93E-04

b. Intra-protocol Fairness

In this experiment, we evaluate the fairness of LTCP flows to each other. Different number of LTCP flows are started at the same time (with random staggering to avoid synchronization) and the average per-flow bandwidth of each flow is noted. Table III shows that when the number of flows is varied, the maximum and the minimum per-

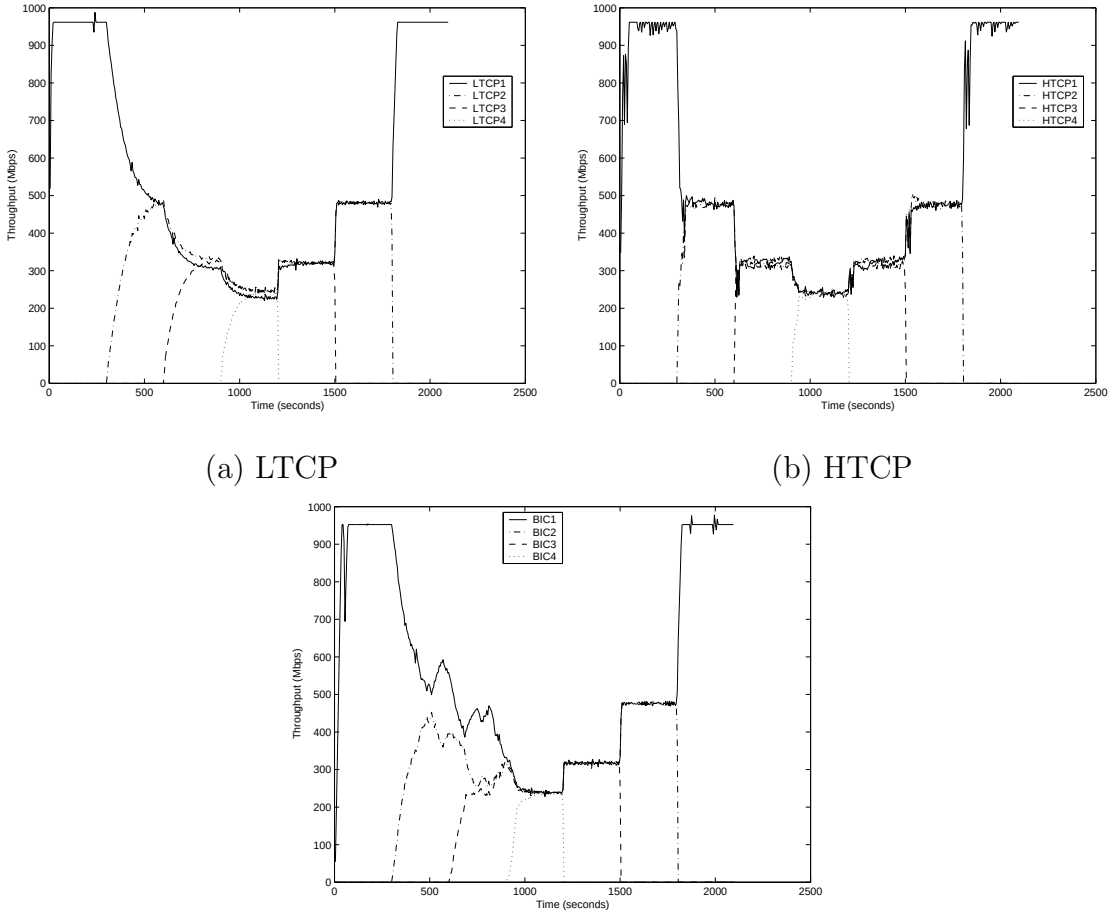
flow throughput remain close to the average, indicating that the per-flow throughput of each flow is close to the fair proportional share. This is verified by calculating the Fairness Index proposed by Jain et. al., in [69]. The Fairness Index being close to 1 shows that the LTCP flows share the available network bandwidth equitably. Similar behavior was observed with BIC and HTCP.

Table III. Fairness Among LTCP Flows

No. of Flows	Avg. per-flow Throughput (Mbps)	Min. per-flow Throughput (Mbps)	Max. per-flow Throughput (Mbps)	Standard Deviation	Jain Fairness Index
2	480.77	480.34	481.20	0.60	1.0000
4	240.39	240.28	240.55	0.12	1.0000
6	160.26	160.14	160.37	0.10	1.0000
8	120.19	120.16	120.31	0.05	1.0000
10	96.15	95.75	99.55	1.19	0.9999

c. Dynamic Link Sharing

In the previous experiment with multiple flows, all the flows were started at about the same time and all different protocol flavors showed very good intra-protocol fairness. In this section, we evaluate the convergence properties when flows start and stop at different times, dynamically changing the available link bandwidth. The first flow is started at time 0, and allowed to reach steady state. A new flow is then added every 300 seconds. The flows last for 2100, 1500, 900 and 300 seconds respectively. Fig. 31 shows the throughput of each flow. From the graph we see that, the BIC flows take much longer to converge to fair share compared to LTCP and HTCP. When bandwidth becomes available because of the completion of a flow, all three protocols are capable of quickly increasing their sending rates and hence ensuring the link is fully utilized.



(a) LTCP

(b) HTCP

(c) BIC

Fig. 31. Dynamic Link Sharing

d. Effect of Random Losses

Fig. 28 shows the response curve of the different high-speed proposals. [60] states that the response function of H-TCP is similar to that of HS-TCP. In this simulation, we show the effect of the different response curves. We fix the capacity of the bottleneck link at 1Gbps and induce random losses on the link using a uniform loss model. Fig. 32 plots the throughput against the random loss rate. Note that the loss rate on the x-axis does not include the self-induced congestion losses. Packet loss rate of 10^{-7} due to channel errors is comparable to the error rate in long haul fiber links [56]. As the random loss rate increases, the link utilization of the different protocols starts to deteriorate. The deterioration of LTCP and BIC are similar, whereas the deterioration of HTCP is slightly more drastic.

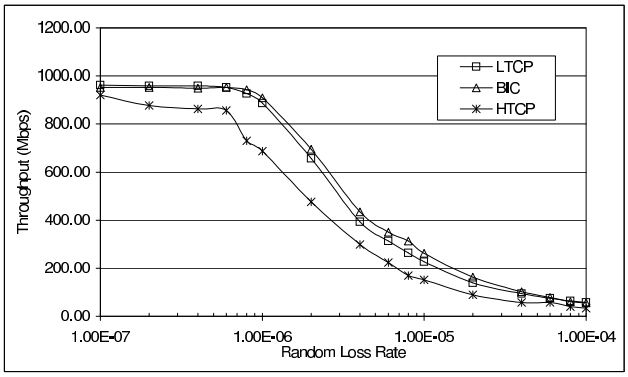


Fig. 32. Effect of Random Losses on Different High-speed Protocols

e. Impact of Bottleneck Link Buffer Size

In this experiment we study the impact of the bottleneck link buffer size on the performance of the different high-speed protocols. For the given topology, the bandwidth-delay product(BDP) is 15000 packets. We vary the bottleneck link buffer size from

1 times to 0.1 times the BDP. Fig. 33 shows the results. As seen from the graph, when the bottleneck link buffer size is at least 0.5 times the BDP, all the high-speed protocols maintain high link utilization. When the bottleneck link buffer size reduces below this, the throughput starts to degrade a little, with the degradation for HTCP being slightly worse than that of BIC or LTCP. As indicated in Section e, even when the buffer size at the bottleneck link router is low, LTCP can maintain high link utilization.

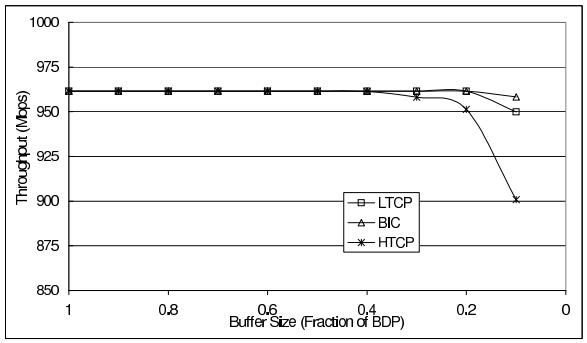


Fig. 33. Impact of Bottleneck Link Buffer Size

f. Interaction with TCP

In this section, we verify the effect of LTCP on regular TCP flows. It must be noted that the window response function of LTCP, BIC and HTCP are *designed* to be more aggressive than TCP in high speed networks. So a single flow of TCP cannot compete with a single flow of these high-speed protocols. We present these results here to show that the high-speed protocols do not starve the TCP flows for bandwidth. Table IV shows the results with one TCP flow sharing the bottleneck link with a high-speed flow. With an increase in the available bottleneck link capacity, the throughput achieved by the TCP flow slightly increases. The throughput obtained by the TCP

flow is similar when competing with the different high-speed flows.

Table IV. Interaction of LTCP with TCP

Bottleneck Link Bandwidth	LTCP		HTCP		BIC	
	TCP Throughput (Mbps)	LTCP Throughput (Mbps)	TCP Throughput (Mbps)	HTCP Throughput (Mbps)	TCP Throughput (Mbps)	BIC Throughput (Mbps)
100Mb	1.45	94.70	3.80	92.29	1.41	93.81
500Mb	4.50	476.27	7.53	471.82	3.77	472.42
1Gb	7.23	954.31	10.35	945.25	6.75	945.63

g. RTT Unfairness

In our analyses we have showed that the RTT unfairness of LTCP can be tuned based on different requirements by modifying K_R. We choose K_R so that LTCP displays RTT unfairness similar to that of TCP, that is, the ratio of the throughput of two LTCP flows with different RTTs is proportional to the inverse square of the ratio of the RTTs. While, the choice of K_R can easily be modified to offer different levels of fairness, we present in this section the results with the chosen design. Each simulation consists of two flows with different RTTs competing for bandwidth on the bottleneck link. The RTT of the shorter link is fixed at 40ms, while varying the RTT of the larger link such that the RTTs have ratio 2, 3 and 4. The bottleneck link capacity is fixed at 1Gbps. Table V shows the results in comparison with HTCP and BIC. Since the scaling factor for HTCP is chosen to provide linear unfairness (as opposed to the square unfairness of TCP), HTCP shows better performance. According to [59], the RTT unfairness of BIC is the same as that of TCP for high bandwidth, and the low RTT flows may starve the high RTT flows at low bandwidth. In this experiment, the RTT unfairness of BIC was observed to be a little worse than that of the inverse square unfairness of TCP.

Table V. RTT Unfairness

RTT Ratio	LTCP	HTCP	BIC
2	3.36	1.85	9.74
3	7.63	2.78	16.61
4	13.65	3.68	26.96

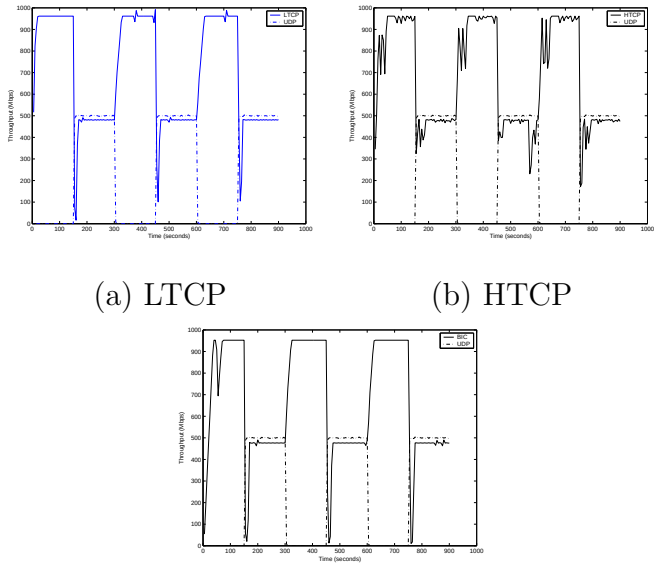
h. Interaction with Non-responsive Traffic

In order to evaluate how LTCP responds to the presence of traffic that does not respond to congestion, we conducted the following simulation. In this simulation, non-responsive on-off traffic was simulated using CBR/UDP source that sends data at half the bottleneck link capacity (500Mbps) for 150 seconds and then remains inactive for the next 150 seconds. Fig. 34 shows the throughput of the two flows computed over 5 second intervals. Results of similar experiments with BIC and HTCP are included for comparison. As seen from the graph, the response of all the three protocols is similar. In the presence of non-responsive traffic all of them reduce their sending rate. When the non-responsive flows are not present, all of them quickly ramp up the sending rate.

5. Emulation Results

We have implemented LTCP in the network stack of the Linux 2.4.25 kernel. The network stack in the 2.4.x kernel is quite sophisticated and supports several standards from the RFCs as well as features beyond those published in RFCs or IETF Drafts aimed to provide good network performance [70].

Our test-bed consists of two off-the shelf Dell Optiplex GX260 workstations with Pentium 4 3.06GHz CPU, 512MB of RAM, Intel PRO/1000 MT gigabit NICs on to a 33MHz/32bit PCI bus. The two computers are connected using a copper Cat 6 cable. The 33MHz PCI bus limits the achievable throughput to around 750-800 Mbps. The



(a) LTCP (b) HTCP

(c) BIC

Fig. 34. Interaction with UDP Traffic

netem network emulator available as part of the *tc* tools is used to generate uniform delay for all packets, so that larger RTTs can be emulated. This patch modifies the FIFO queue such that every packet queued is delayed for a fixed amount. This setup was used instead of the conventional sender-router-receiver setup, due to the limitation imposed by the PCI bus which in the router configuration would further reduce the bottleneck link capacity as it is shared by both incoming and outgoing interfaces. *iperf* [71] was used for generating traffic. We increased the socket buffers to allow maximum link utilization. The txqueuelen for the NIC was set to the default value of 1000. The backlog queue was modified to have a size of 1000.

In all experiments we show results for three RTT values - 25ms, 70ms and 150ms - which are representative of links across close by cities, across the country and transatlantic links. These values are chosen based on the pingER measurement history tables at SLAC [72].

a. Basic Performance Tests

We present here the results of the experiment comparing the performance of the standard Linux TCP (TCP-SACK) with that of LTCP at different RTTs for a 900 second transfer. The socket buffer size at both the sender and receiver is set to 32MB to ensure that a single flow can utilize all the available bandwidth. The experiment is run for 15 minutes (900 seconds) and is repeated four times. The table in Table VI shows the average number of bytes transferred and the average transfer rate. At low RTTs, both the TCP and LTCP flows manage to keep the link almost fully utilized and transfer about 75 GBytes of data. As the RTT increases, TCP takes longer to recover from packet losses and its performance starts to deteriorate. For an RTT of 120ms, which is comparable to that of the transatlantic links, the performance deterioration is significant and in 900 seconds, the TCP flow manages to transfer only about 22 GBytes. In contrast, a single LTCP flow transfers an average of about 65 GBytes.

Table VI. Linux Performance Test

	Total Bytes (GBytes)				Rate (Mbits/s)			
	TCP		LTCP		TCP		LTCP	
	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation
Low RTT	75.40	0.08	77.85	0.06	719.25	0.50	742.75	0.50
Moderate RTT	50.33	5.73	71.38	0.55	480.25	54.52	681.50	5.00
High RTT	17.20	1.47	48.85	1.24	164.00	14.02	466.00	11.86

b. Performance with Fixed Transfer Size

The above experiment illustrates the effectiveness of the LTCP protocol in maintaining higher link utilization than TCP. It is based on the assumption that the application has an infinite amount of data to send and hence only one connection

establishment is required for the entire 15 minute period. This assumption may not always be true. In this experiment, we find the time it takes to transfer 100 GBytes of data in chunks of 2 GBytes each. A new transfer is started after the completion of the previous transfer. This experiment captures the slowstart and connection tear down dynamics. Table VII shows the results. Again at low RTT values, the difference is not noticeable. However, at larger RTTs, the LTCP protocol can transfer the data in less than half the time of TCP. Table VIII shows the statistics for the individual transfers of 2 GByte size. From the table it is clear that the standard deviation of both the time to transfer and the rate of transfer is larger for the LTCP flows. Recollect that LTCP does not modify the behavior during the slowstart period. However, when a flow comes out of slowstart, the number of layers (K) is updated according to the congestion window size. Hence, depending on when the slow start terminates, the different flows would operate at different layers. Since each transfer is short and slow start controls a large part of the transfer, sufficient time is not available for the LTCP flows to reach the stable operating point. Hence the increased variance in the time and the rate of transfer.

Table VII. Time for Transferring 100 GBytes of Data in Units of 2 GBytes

	TCP	LTCP
Low RTT	19 min 09 Sec	19 min 09 Sec
Moderate RTT	1 hour 01 min 36 Sec	31 min 08 Sec
High RTT	2 hour 22 min 22 Sec	1 hour 09 min 15 Sec

c. Fairness Among Multiple Flows

In order to evaluate the inter-protocol fairness among multiple flows of LTCP w e conducted the experiment with multiple flows starting at the same time and calculated the Jain Fairness Index [69]. Table IX shows the results. The Jain Fairness Index for

Table VIII. Statistics for Individual 2 GByte Transfers

	Time (Seconds)				Rate (Mbits/s)			
	TCP		LTCP		TCP		LTCP	
	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation
Low RTT	22.950	0.051	22.966	0.048	730.980	0.553	730.780	1.016
Moderate RTT	73.852	0.231	37.298	7.445	227.060	0.740	468.040	92.815
High RTT	170.690	0.486	82.942	18.222	98.292	0.269	212.300	46.563

LTCP is high across different RTTs and different number of flows. Also, the standard deviation in the per-flow throughput of the LTCP flow is fairly low. This indicates that multiple flows of LTCP protocol share the available bandwidth equitably. Results for TCP are included to aid the comparison.

Table IX. Fairness Among Multiple Flows

RTT	Number of flows	TCP			LTCP		
		Average per-flow goodput (Mbits/s)	Standard Deviation	Jain Fairness Index	Average per-flow goodput (Mbits/s)	Standard Deviation	Jain Fairness Index
Low RTT	2	373.50	50.20	0.991	373.00	12.73	0.999
	4	186.25	24.98	0.987	186.50	8.81	0.998
	6	124.00	12.63	0.991	124.17	2.32	1.000
	8	92.93	3.26	0.999	92.96	2.05	1.000
	10	74.26	4.13	0.997	74.28	2.69	0.999
Moderate RTT	2	299.50	64.35	0.977	353.50	9.19	1.000
	4	175.50	12.01	0.996	181.75	4.86	0.999
	6	116.77	20.40	0.975	121.50	6.53	0.998
	8	89.35	23.03	0.945	91.31	1.90	1.000
	10	70.53	10.33	0.981	73.04	4.66	0.996
High RTT	2	128.50	37.48	0.959	286.50	26.16	0.996
	4	123.73	30.21	0.957	166.00	7.53	0.998
	6	103.85	17.98	0.976	113.62	16.58	0.983
	8	78.06	39.80	0.815	86.68	8.72	0.991
	10	66.06	21.02	0.916	69.61	6.76	0.992

d. Interaction of LTCP with UDP-based Traffic

In order to evaluate how LTCP responds to the presence of dynamically changing traffic that does not respond to congestion, we conducted the following experiment. In this experiment, an iperf stream of UDP sending at 350Mbps (about half the available link capacity) at intervals of 5 minutes was used as the source of non-responsive on-off traffic. UDP traffic consumes a lot of resources on the machines.

In order to offset this, the txqueuelen and the backlog queue were set to 5000 in this experiment. Fig. 35 shows the results. It is clear from the graphs, that when the UDP stream is sending packets, the LTCP source reduces its congestion window so that the link is shared between the LTCP and the UDP traffic. When the UDP source, stops sending packets, the LTCP flow increases its sending rate close to the full link capacity. We verified this at different RTTs, but have included the results here for the medium RTT case.

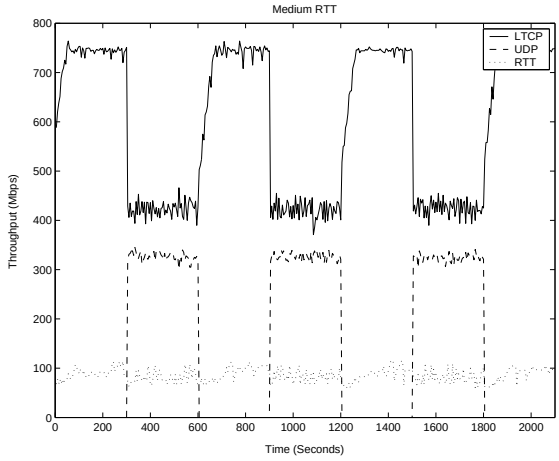


Fig. 35. Interaction of LTCP with Non-responsive Traffic

B. LTCP with Variable β

The LTCP variant discussed in the previous section used a fixed window decrease factor (β) of 0.15, which imposed an upper limit of 19 on the number of layers (K). In this section we remove this restriction by making β adaptive, such that the window decrease at any given layer is β_K. The value for β_K is chosen such that conditions in Eq. 3.2 and 3.3 are satisfied and the fairness properties are retained.

1. Analysis

For determining the appropriate value for β_K, we start with the decrease behavior and analyze the behavior for flows with similar RTTs, similar to Eq. 3.4. The decrease behavior is again chosen to be multiplicative - however, the decrease factor now depends on the layer at which the flow operates. As a result, we have -

$$\omega = \beta_K * W \tag{3.29}$$

Similar to the discussion before, based on this choice for the decrease behavior, we determine the appropriate increase behavior such that the conditions in Eq. 3.2 and 3.3 are satisfied.

Consider Eq. 3.3

$$\frac{\omega_1}{K_1^{'}} > \frac{\omega_2}{K_2^{'}} \tag{3.30}$$

If we continue to have the same stipulation as before that at most one layer can be dropped after a packet loss, then it is most difficult to maintain the convergence properties, when the larger flow does not reduce a layer but the smaller flow does, i.e.,, $K_1^{'} = K_1, K_2^{'} = (K_2 - 1)$. With this worst case situation, Eq. 3.3 can be written

as -

$$\frac{\omega_1}{K_1} > \frac{\omega_2}{(K_2 - 1)} \tag{3.31}$$

If this inequality is maintained for adjacent layers, we can show by simple extension, that it can be maintained for all other layers. So consider $K_1 = K, K_2 = (K-1)$. Then, the above inequality is

$$\frac{\omega_1}{K} > \frac{\omega_2}{K - 2} \tag{3.32}$$

Suppose, the window for flow 1 is $W^{'}$ when the packet loss occurs and the window of flow 2 is $W^{''}$ then, substituting Eq. 3.29 in the above equation, we have,

$$\begin{aligned} \frac{\beta_K * W^{'}}{K} &> \frac{\beta_{K-1} * W^{''}}{K - 2} \\ \Rightarrow W^{'} &> \frac{K}{K-2}\frac{\beta_{K-1}}{\beta_K} W^{''} \end{aligned} \tag{3.33}$$

In order for the worst case behavior ($K_1^{'} = K, K_2^{'} = (K-2)$) to occur, the window $W^{'}$ could be close to the transition to the layer $(K+1)$ and the window $W^{''}$ could have recently transitioned into layer $(K-1)$. In order to get the estimate of the worst case, we substitute these values in the above equation to get

$$W_{K+1} > \frac{K}{K-2}\frac{\beta_{K-1}}{\beta_K} W_{K-1} \tag{3.34}$$

Based on this, we choose the increase behavior to be

$$W_K = \frac{K}{K-2}\frac{\beta_{K-1}}{\beta_K} W_{K-1} \tag{3.35}$$

With this choice the equations for W_K and δ_K remain similar to those in Eq. 3.11 and 3.12 as shown below.

Consider first the choice of β_K. Based on our stipulation that at most one layer

may be dropped after a window reduction, we have (similar to Eq. 3.13)

$$\beta_K W_K < \delta_{K-1} \tag{3.36}$$

Suppose we choose $\beta_K = \alpha\frac{\delta_{K-1}}{W_K}$ where $\alpha < 1$. Substituting this in Eq. 3.35 we have,

$$\begin{aligned} W_K &= \frac{K}{K-2}\frac{\frac{\alpha\delta_{K-2}}{W_{K-1}}}{\frac{\alpha\delta_{K-1}}{W_K}}W_{K-1} \\ &\Rightarrow \frac{\delta_{K-1}}{\delta_{K-2}} = \frac{K}{K-2} \\ &\Rightarrow \frac{\delta_K}{\delta_{K-1}} = \frac{K+1}{K-1} \end{aligned} \tag{3.37}$$

Suppose, just as before, we start adding layers when window is equal to W_T. The we have $\delta_1 = W_T$. By recursive substitution in the above equation we have,

$$\delta_K = \frac{K(K+1)}{2}W_T \tag{3.38}$$

which is similar to the earlier design with fixed β.

Now consider W_K. Since we have layer threshold W_2 at W_T and a new layer is added when window increases by δ_K we have,

$$W_K = \delta_1 + \delta_2 + \ldots + \delta_{K-1} \tag{3.39}$$

By simple summation of the above equation, we get

$$W_K = \frac{K(K+1)(K-1)}{6}W_T \tag{3.40}$$

which is again similar to the earlier design with fixed β.

Hence, we now have a design for LTCP with variable β while maintaining basic equations for W_K and δ_K similar to the earlier design with fixed K. Note that these equations are not effected by the exact value of β_K, as long as it is chosen based on the constraint that $\beta_K = \alpha\frac{\delta_{K-1}}{W_K}$ where $\alpha < 1$. In order to determine the appropriate value for α consider the following :

$$\beta_K = \alpha\frac{\delta_{K-1}}{W_K}$$
$$\Rightarrow \beta_K = \alpha\frac{\frac{(K-1)K}{2}W_T}{\frac{K(K+1)(K-1)}{6}W_T} = \alpha\frac{3}{K+1} \tag{3.41}$$

When $K = 1$, we would prefer to maintain the behavior of LTCP similar to that of standard TCP and so we choose $\beta_1 = 0.5$. This gives us the value of α to be $\frac{1}{3}$. Note that, even though the analysis above has been conducted with discrete values for beta, in the implementation, we use a continuous linearly scaled value similar to the scaling used for increase parameter K in Section III.A.3.f.

Since W_K and δ_K with the new design is similar to that of the previous design, the time to claim bandwidth remains similar to that in Table I. However, since the window reduction β_K is now a variable, time to recover from packet loss is different. Fig. X shows the relationship of layers K to the window thresholds W_K and window reduction factors β_K and also the difference in the speedup in packet loss recovery time compared to Table I.

Note that with the choice of variable β, there is no longer a restriction on number of layers that a flow can grow up to and LTCP is scalable with increasing bandwidth. Also, with the new design, the speedup in packet loss recovery time is smaller at smaller layers, but higher in larger layers.

Table X. Comparison of LTCP (with $W_T = 50$ and variable β) to TCP

K	W_K	β_K	Speedup in Claiming Bandwidth	Speedup in Packet Loss Recovery Time
1	0	0.5000	-	-
2	50	0.3333	1.00	1.50
3	200	0.2500	2.00	4.00
4	500	0.2000	2.57	7.50
5	1000	0.1667	3.17	12.00
6	1750	0.1429	3.78	17.50
7	2800	0.1250	4.40	24.00
8	4200	0.1111	5.03	31.50
9	6000	0.1000	5.67	40.00
10	8250	0.0909	6.31	49.50
11	11000	0.0833	6.95	60.00
12	14300	0.0769	7.60	71.50
13	18200	0.0714	8.25	84.00
14	22750	0.0667	8.90	97.50
15	28000	0.0625	9.56	112.00
16	34000	0.0588	10.21	127.50
17	40800	0.0556	10.87	144.00
18	48450	0.0526	11.52	161.50
19	57000	0.0500	12.18	180.00
20	66500	0.0476	12.84	199.50
21	77000	0.0455	13.50	220.00
22	88550	0.0435	14.16	241.50
23	101200	0.0417	14.82	264.00
24	115000	0.0400	15.48	287.50
25	130000	0.0385	16.14	312.00

2. NS-2 Simulation Results

In order to demonstrate that using variable β does not modify the basic behavior of LTCP, we have repeated some of the experiments whose results were presented earlier. We show the results below.

a. Window Behavior, Link Utilization and Packet Loss Rates

Fig. 36 shows congestion window of the LTCP protocol with variable β. The operating window size is around 19000 packets, which corresponds to layer number 13 and the β_K value is 0.071 as seen from Table X. Thus, the window decrease after a packet loss is slightly lower than the fixed β version and hence the oscillation of the congestion window about the optimal point is slightly smaller.

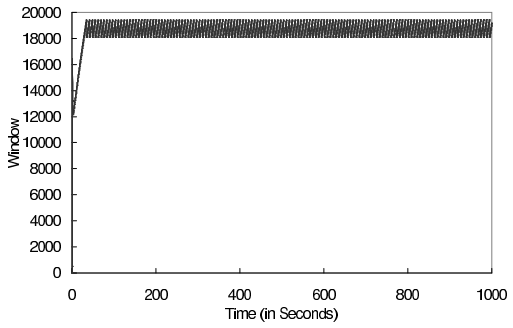


Fig. 36. Congestion Window Behavior of LTCP with Variable β

Table XI shows the throughput and drop rates of for LTCP with variable β. The results for LTCP with fixed β are included for comparison. As seen from the table, the link utilization remains similar, but the drop rates are slightly higher when variable β is used, especially at higher bottleneck link capacities where the window can get large. At larger windows, the value of β_K gets smaller and hence the window

oscillates close to the optimal operating point. Since each oscillation results in a congestion event, the congestion droprate is slightly higher.

Table XI. Goodput and Packet Loss Rate Comparison for LTCP with Fixed and Variable β

Bottleneck Link Bandwidth	LTCP with Fixed β		LTCP with Variable β	
	Goodput (Mbps)	Packet Loss Rate (%)	Goodput (Mbps)	Packet Loss Rate (%)
100Mb	96.15	4.07E-02	96.20	4.34E-02
250Mb	240.38	1.21E-02	240.42	1.64E-02
500Mb	480.77	4.83E-03	480.81	7.97E-03
1Gb	961.54	1.95E-03	961.58	3.97E-03
2.4Gb	2307.69	6.10E-04	2307.74	1.63E-03

b. Intra-protocol Fairness

Next we verify that using variable β does not negatively impact the fairness behavior of LTCP. Similar to the experiment before, several LTCP flows share the bottleneck link capacity. Table XII shows the results. With variable β, the standard deviation amongst the average throughput of the individual flows increases slightly. However, the Jain Fairness Index remains high, indicating that the the flows remain fair to each other.

Table XII. Fairness Among LTCP Flows with Variable β

No. of Flows	Avg. per-flow Throughput (Mbps)	Min. per-flow Throughput (Mbps)	Max. per-flow Throughput (Mbps)	Standard Deviation	Jain Fairness Index
2	480.61	470.38	490.84	14.47	0.9995
4	240.30	232.27	254.68	10.62	0.9985
6	160.20	145.91	177.25	11.32	0.9959
8	120.15	115.77	130.83	5.19	0.9984
10	96.12	88.84	100.36	4.31	0.9982

c. Interaction with TCP

As discussed earlier, the high-speed protocols are by design more aggressive than TCP and as such cannot be expected to co-exist with TCP. In this experiment, we verify that, while LTCP with variable β gains higher throughput than TCP, it does not starve TCP in the process. The experiment is similar to the experiment before. The bottleneck link is shared by two flows, one using LTCP with variable β and the using TCP-SACK. Table XIII shows the results. The results for LTCP with fixed β are included for comparison. From the figure we see that the throughput of the TCP flow is slightly higher when competing with LTCP using variable β. This is because, at lower layers the value of variable β is higher than the fixed value used before, allowing TCP to grab some additional bandwidth after a packet loss.

Table XIII. Interaction of LTCP (using variable β) with TCP

Bottleneck Link Bandwidth	LTCP with Fixed β		LTCP with Variable β	
	TCP Throughput (Mbps)	LTCP Throughput (Mbps)	TCP Throughput (Mbps)	LTCP Throughput (Mbps)
100Mb	1.45	94.70	2.36	92.08
500Mb	4.50	476.27	6.81	468.55
1Gb	7.23	954.31	17.98	935.31

d. RTT Unfairness

Lastly, we repeat the RTT unfairness experiment from earlier where two flows with different RTTs using LTCP share the bottleneck link. The RTT of the smaller RTT flow is 40ms. The other flow has RTTs of 80ms, 120ms and 160ms in different runs of the experiment, resulting in the RTT ratios of 2, 3 and 4. For the LTCP protocol using variable β we use the same RTT compensation factor K_R as we used with LTCP with fixed β. Table XIV shows the results. From the table, we see that for LTCP

with variable β the inverse throughput ratio is similar to the RTT ratio rather than the square or the RTT ratio displayed by TCP and LTCP with fixed β.

Table XIV. RTT Unfairness of LTCP Flows with Variable β

RTT Ratio	LTCP with Fixed β	LTCP with Variable β
2	3.36	2.43
3	7.63	4.05
4	13.65	4.18

More experimental evaluation of the LTCP protocol using variable β is presented in Section V.C.

C. Conclusions

In this chapter, we proposed LTCP, a layered approach for modifying TCP for high-speed links. LTCP employs two dimensional control for adapting to available bandwidth. At the macroscopic level, LTCP uses the concept of layering to increase the congestion window when congestion is not observed over an extended period of time. Within a layer K, LTCP uses modified additive increase (by K per RTT) and remains ACK-clocked. The layered architecture provides flexibility in choosing the sizes of the layers for achieving different goals. In this chapter we have considered one possible design and presented two options for the window decrease function - fixed decrease and adaptive decrease.

We have shown through analysis, simulations and emulations that a single LTCP flow can adapt to nearly fully utilizing the link bandwidth. Other significant features of the chosen design are - (a) it provides a significant speedup in claiming bandwidth and in packet loss recovery times (b) multiple flows share the available link capacity equitably (c) it could potentially alleviate the RTT unfairness inherent to TCP (d) requires simple modifications to TCP's congestion response mechanisms.

CHAPTER IV

USING TCP FOR CONGESTION AVOIDANCE INSTEAD OF CONGESTION CONTROL

The basic mechanism used in TCP recognizes two states - "congestion" or "no congestion". A packet loss is implicitly assumed to indicate "congestion". Lack of packet loss within a window is implicitly assumed to indicate "no congestion". As a result, even when the bottleneck link is saturated and the router buffers start to fill up, as long as there are no packet losses, TCP continues to increase its congestion window (and hence the resulting sending rate). This inability of TCP to recognize the onset of congestion results in it contributing to an increase in the congestion buildup.

The need for early response to the onset of congestion in the Internet - i.e., to have the flows back off from the active probing mode when the queues start to fill up - has long been recognized and has been an area of active research for several years. Two rivaling approaches have been suggested for addressing the problem. One school of thought has been that the routers at the bottleneck link know when the buffers are getting full and so are in the best position to judge the onset of congestion. Once congestion is detected, they inform the end-host implicitly by either a binary signal (e.g., RED[73], REM[74], PI[75], AVQ[76], BLUE[77], etc.) or a non-binary signal which informs the senders of what their sending rate should be (e.g., XCP[61], RCP[78], VCP[79] and JetMax[80], etc). This is the area broadly known as Active Queue Management.

The other school of thought treats the network as a black box and tries to detect network congestion at end-hosts. In these schemes, the sender tries to determine the level of congestion in the network based on information contained in the round trip time or the observed throughput and responds to it in an effort to prevent packet

losses. In this approach, the senders do not expect any support from routers. We refer to the work in this area as end-host delay-based congestion avoidance. Some of the representative schemes in this area are CARD[81], TRI-S[82], DUAL[83], Vegas[58] and CIM[84] among others.

Both the mechanisms have their advantages and disadvantages. While it is true that the routers are in the best position to detect congestion in the network, deploying experimental AQM mechanisms in the core routers has not been easy. Overcoming the technical challenges related to the implementation of these schemes in the routers, while retaining the speed and efficiency at which they operate, is probably an easier task compared to convincing the operators of ISP's to replace or add these experimental routers to production networks. Comparatively, updating TCP in the network stack at end-hosts is a relatively easier task. But end-host delay based schemes have been assailed by doubts about the accuracy and effectiveness of the congestion prediction at end-hosts [84, 85, 86].

In this chapter, we focus on evaluating and improving end-host based congestion prediction. We show that end-host based congestion prediction is more accurate than characterized by studies in [84, 86]. While it is possible to further improve the prediction ability of the end-host based congestion estimators, it may not be possible to entirely eliminate the noise/uncertainty in the signal. We show that this uncertainty can be offset by choosing an appropriate response function. By emulating the probabilistic marking mechanism of the router-based AQM schemes in the response function of the end-hosts, performance and benefits similar to router-based AQM can be obtained *without actually requiring any modifications to the routers*. We support our claim via extensive simulations on the ns-2 simulator. While we have chosen to illustrate the benefits of the scheme by emulating RED/ECN [73], it is possible to emulate other router based schemes as well. To illustrate this, we present some

preliminary results for the emulation of REM/ECN [74], as well.

The rest of the chapter is organized as follows. Section A inspects the existing end-host based congestion prediction signals and some of the studies that cast doubts about their accuracy. We show why some of the claims made by these measurement studies may be inaccurate. We then proceed to determine how to further improve the prediction efficiency of end-host based estimators, and based on our observations, we determine the appropriate signal that we will use in our proposed scheme. In Section B we design the response function that emulates RED/ECN behavior. Section C presents an extensive evaluation using ns-2 simulations over a wide range of network conditions. Here, we also examine the flexibility of the scheme to show that other AQM techniques can also be emulated, by providing some preliminary results of the emulation of REM/ECN. In Section D we offer a critical discussion of the issues that are still open and the possible directions for future work in this area. We conclude the paper in Section E.

A. Congestion Detection at End-hosts

Over the years, as TCP evolved, the line between the definition of the terms *congestion avoidance* and *congestion control* has blurred. The Additive Increase/Multiplicative Decrease algorithms of TCP, together with fast retransmit/recovery are commonly referred as the congestion avoidance algorithms. It must be noted however, that these algorithms are not designed to proactively avoid congestion and the current dominant versions of TCP, say TCP-SACK or TCP-Newreno, are strictly *congestion control* schemes. These schemes reduce the congestion window and hence the prevailing congestion only *after* the fact that a packet loss has occurred. Parallel to these algorithms, proactive schemes which focus on congestion avoidance have also evolved.

These schemes use information contained in the round trip time or the observed throughput of the flow to predict congestion in the network. Based on this prediction, they aim to reduce the congestion window even *before* a packet loss can occur, with an intent to reduce (and possibly avoid) the packet losses.

In 1989, Raj Jain [81] first proposed enhancing the congestion control algorithm of TCP by adding a delay-based congestion avoidance mechanism. The main observation is that, as the sending rate of a flow is increased so does its achieved throughput, until the knee where link is nearly saturated. During this time, the delay observed by the flow will be low. Beyond the knee point, the throughput stabilizes and the delay increases sharply. The author proposed monitoring the normalized delay gradient of a flow at the sender for determining the knee point and making the flow operate at this point. Ever since this proposal, several different schemes have appeared. In TRI-S [82] the authors propose using the normalized throughput gradient instead of the normalized delay gradient to foresee when the bottleneck link reaches saturation. In DUAL [83] the authors compare the current sample of RTT to the average of the minimum and maximum RTT to determine that the bottleneck link queues are more than half full and it is time to respond. In Vegas [58] the authors propose comparing the achieved throughput to the expected throughput based on minimum observed RTT for predicting congestion. In CIM [84] the authors propose comparing the moving average of a small number of RTT samples to the moving average of a large number of RTT samples for determining if there is any congestion. In TCP-BFA [87] the authors propose monitoring the variance of RTT for avoiding the bottleneck link from filling up. In Sync-TCP [88] the authors use the trend of one-way delays combined with four different levels of window increase/decrease to improve on delay-based congestion avoidance.

Several other studies [85, 84, 86, 89] have focused on understanding whether

delay-based congestion prediction can reliably work in the real Internet. While some of the issues raised by these studies are unique to end-host based mechanisms, some of the issues have received similar attention in the router based schemes. For instance, it has been pointed that it is hard to measure queuing delays accurately when they are very small compared to the end-to-end RTT [85, 84, 86]. However, this is a limitation of not just end-host based schemes but also router-based schemes, since a large RTT compared to the queue length implies a large feedback loop. In such cases, the router queue may fill up and result in a loss before the sender receives and responds to this congestion feedback. End-host based measurements of the state of the queue essentially entails sampling the queue at certain times and the fundamental limits of such measurements have been recently highlighted [90, 86]. Problems of oversampling the queue lengths in router based RED mechanisms have been studied in [75]. It has been suggested that on highly aggregated paths, the impact of the response of a single flow may be limited [89, 84]. But if several flows respond, then the combined effect may be sufficient to relieve the congestion and reduce the queue lengths. It must be noted that the aim of the congestion avoidance schemes is not to replace the congestion control mechanism based on packet losses but to enhance it by providing an additional mechanism for detecting congestion.

The focus of this study, is to understand the issues specific to end-host based schemes. In [84] the authors have measured the correlation between the observed loss rate of the flow and the observed round trip time samples just before the loss for data collected using tcpdump on seven Internet paths. Their observation is (a)the losses are preceded by a significant increase in RTT in very few cases (b)responding to a wrong prediction can result in severe degradation of performance. Based on their observations the authors conclude that although the RTT samples may contain useful information, it cannot be reliably used. Another study [86] shows similar results but

over a larger dataset and considers several of the different delay-based prediction metrics mentioned at the beginning of this section.

In order to understand these claims better, consider a very simple scheme that directly monitors the observed round trip delay. Fig. 37 shows a simple state diagram of the different states and transitions associated with end-host congestion control. State A represents the "low delay" or "low congestion" state. The flow transitions from state A to B, The first state transition marked "1" will take the flow from state A to state B, when higher delay is observed, or the bottleneck queue is starting to get filled. If no action is taken, then state transition marked "2" will occur taking the flow to state C or the "packet loss" state. In response to the packet loss, when the flow reduces its congestion window, the bottleneck queue starts to empty out and the flow will eventually return to State A via transition "3". Now consider the other possible state transitions "4", "5" and "6".

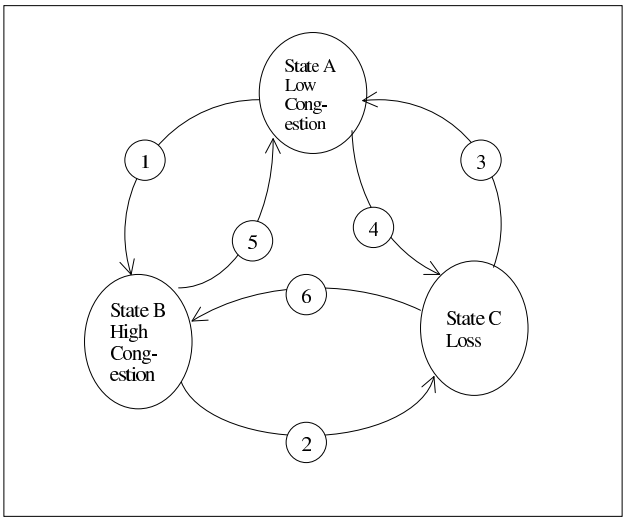


Fig. 37. State Diagram Showing the Transition between Different Congestion States in Standard TCP Flows

If the flow does not use an accurate method for determining the states, or if the

flow shares the bottleneck link with a lot of cross traffic, then the duration of state B may be too short and the flow may not be able to detect it at all. As a result, before the flow can detect that the delay has increased, it may observe a packet loss. This is transition "4" from state A, directly into state C. Alternately, the transitions are governed by how the protocol defines the state A and state B and what criteria is used for transition "1" from state A to state B. If the protocol is very aggressive, then it may determine the onset of congestion too early and make the transition "1" from state A to state B, only to find out later that it was a false alarm and return from state B to state A using transition "5". Transition "6" from state C back into state B occurs if the flow does not respond enough after a packet loss.

Both transitions "4" and "5" are harmful to a proactive scheme. Transition "4", may mean that short term congestion has occurred and the flow is not able to predict it and hence is incapable of avoiding the resulting loss or that the protocol is simply not aggressive enough in predicting congestion. This transition indicates the presence of "false negatives". Transition "5", on the other hand, which we will refer to as "false positive", will mean that the protocol is too aggressive or unreliable in predicting congestion, and as a result may unnecessarily reduce its sending rate and face performance degradation.

The claim made in [84, 86] is that for the congestion predictor used in existing schemes, transition "5" happens more often than transition "2" hence limiting the effectiveness of congestion prediction. The authors arrive at this conclusion by looking at a large dataset of *tcpdump* data of standard TCP flows on the Internet. They then run the algorithms used by the different congestion predictors to determine states A and B. The limitation of these studies, however, is that state C, or losses, is observed *within a single flow*. When several different flows share a bottleneck link, the flow under observation may not necessarily be the one to face losses when the bottleneck

link is full. Hence observing high RTT that is not followed by a loss (at a single flow) does not necessarily mean that there is no congestion. It just means that the observed flow does not suffer a packet loss (possibly because other flows have responded to losses and reduced the congestion). In order to evaluate the usefulness of a congestion prediction metric, we should consider the correlation between the round trip time and losses *at the bottleneck link*.

In order to illustrate this, we present here the results of a very simple ns-2 simulation. The topology consists of two routers connected by a link of capacity 100Mbps and delay 20ms. Several nodes are connected to both the routers with links of capacity 500Mbps and varying delay, resulting in different flows having different RTTs. The capacity of the queue at the routers is set to 750 packets. We use six test case loads denoted as case1 through case6 corresponding to the six possible combinations of [50,100] long term flows (in both directions) and [100, 500, 1000] web sessions. One of the long-term flows is tagged and referred as "observed" flow. For this flow, we collect the RTT samples for every packet over a period of 1000 seconds. The end-to-end delay of this flow is 60ms. Using a simple threshold of 65ms to indicate a state of high-RTT, we measure the fraction of transitions from the high-RTT state into the loss state, with losses measured at both the flow level and the queue level. Fig. 38 shows the comparison for the six different cases mentioned above.

While this study is not exhaustive, it clearly indicates that the correlation between RTT and losses observed at the queue are significantly higher than the correlation between RTT and losses observed by a single flow. In other words, delay-based indicators may be a lot more effective in predicting congestion than what is reported in [84, 86].

In order to understand the reliability of the congestion predictors used in different

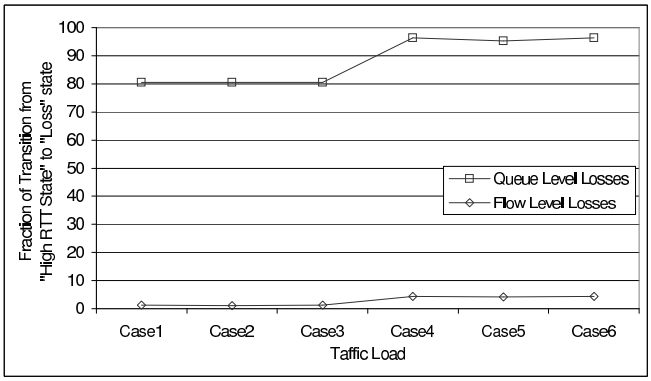


Fig. 38. Comparison of the Fraction of Transitions from "High RTT" to "Loss" When the Losses Are Measured (a) within a Flow and (b) at the Bottleneck Queue.

end-host based schemes, we applied the algorithms used by these schemes for predicting congestion on the data obtained for the six traffic cases above and evaluated the efficiency of predicting losses (at the bottleneck link queue), and the fraction of the false positives and false negatives. We measure the efficiency of loss prediction by the fraction (number of "2" transitions)/("2" transitions + "5" transitions). We measure the false positives as (number of "5" transitions)/("2" transitions + "5" transitions). Finally, we measure false negatives as (number of "4" transitions)/("2" transition + "4" transitions). Fig. 39 shows the results.

From the figure we see that among the existing schemes for end-host congestion prediction, Vegas has the highest prediction efficiency (which implicitly implies the lowest false positives) and the lowest false negatives. Note that these results are not meant to be an exhaustive evaluation of the merits of the different metrics. Rather, it is meant to guide us in our choice for a predictor. From the results, we notice that while the accuracy of congestion prediction may be higher than characterized by earlier measurement studies, it still leaves room for further improvement.

Note that, Vegas, CARD, TRI-S and DUAL, obtain RTT samples once per RTT

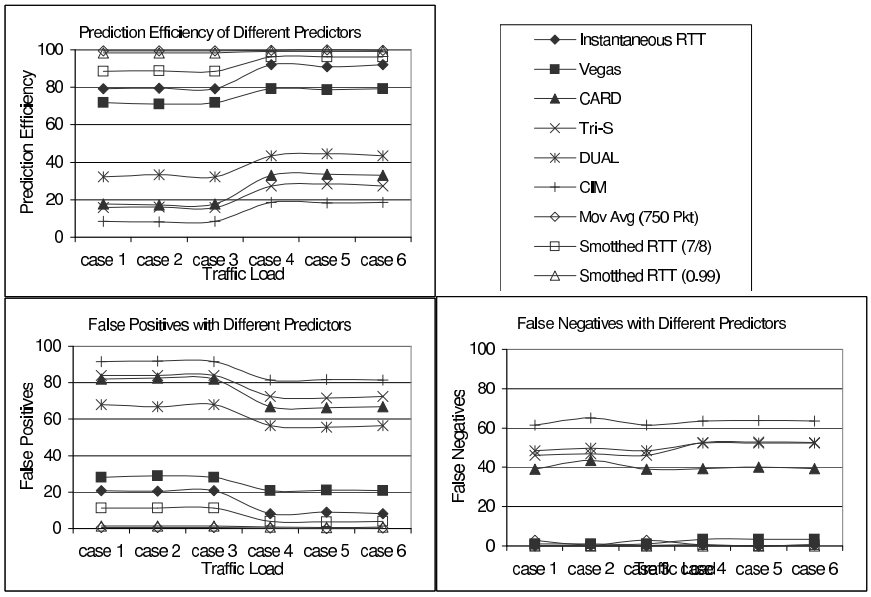


Fig. 39. Prediction Efficiency, False Positives and False Negatives for Different Predictors

which could result in under-sampling. In order to evaluate the impact of higher sampling, we computed the instantaneous RTT[1] upon the receipt of each acknowledgment and used a simple fixed threshold for determining that the flow is in high congestion state. Taking RTT samples on each packet addresses some of the concerns raised about end-host measurements [90] and reduces sampling errors. Surprisingly, as shown in the graph, the prediction efficiency of this signal was higher than that of Vegas for the six test cases of the traffic that we have considered.

While the instantaneous RTT signal is more aggressive in predicting losses, it can be quite noisy due to the fluctuations in the instantaneous queuing delays observed by the packets, and hence false positives are still quite high. In order to eliminate the noise, we smoothed the RTT signal obtained from each acknowledgment. Since

[1]Current versions of the Linux operating systems (2.4.x and above) do this for RTO calculation anyway[70].

the aim of the prediction signal is to track changes in the bottleneck queue, we used a moving average of 750 packets (the size of the buffer) for smoothing the signal. This signal was very effective in predicting losses while avoiding false positives and false negatives. However, it is difficult for a flow to estimate the number of buffers in the bottleneck link. So we investigated the effectiveness of using Exponentially Weighted Moving Average (EWMA) of the instantaneous RTT signal, similar to that used by TCP for determining the retransmission timeout with a weight of 7/8 for the history sample. As seen from the graph, while this reduces the noise (false positives) compared to the instantaneous RTT signal, it still is not as good as the moving average signal. We repeated the EWMA with a higher weight of 0.99 for the history and the signal was able to obtain high prediction efficiency while maintaining low false positives and low false negatives.

We use the smoothed RTT signal with the weight of 0.99 for the history sample, as the congestion predictor in our scheme. In order to differentiate the srtt signal that we use from that used by TCP for timeout calculations, we refer to our signal as $srtt_{0.99}$. Even though the weights used for smoothing are different, note that this signal achieves similar behavior to that in RED routers where average queue lengths are monitored to infer congestion, while accommodating bursty traffic by allowing fluctuations in the instantaneous queue length.

With the $srtt_{0.99}$ signal, while the false positives are low, they are still non-zero - for the six test cases we considered, the false positives were in the range $0.7 - 1.5\%$. This number may vary for other traffic cases. The throughput of a TCP flow is proportional to $\frac{1}{\sqrt{p}}$ (p here will be the probability of response). Responding to even the small fraction of false positives may result in severely degrading the performance as shown in [84]. In this paper, we take a two-step approach: (1) first, we have shown how to improve the accuracy of prediction by using more frequent samples and history

information, and (2) we accept that end-host prediction cannot be perfect and devise mechanisms to counter/mitigate this inaccuracy. In the next section, we discuss the design of the response function.

B. Response to Congestion Prediction at End-hosts

Before we discuss mitigating the impact of false positives, we take a step back and look at the state diagram in Fig. 37 again. Fig. 40 shows the new state diagram incorporating the end-host response. When the flow is in state B (or high congestion state), based on some criteria it moves to state B^1 or the response state with the transition marked "7". If the response is sufficient, then the flow may make transition "5" to state A. If the response is insufficient, the flow may get back into state B using the transition "8". Note that these transitions happen, irrespective of whether the flow is in state B due to a false positive or not.

One possible mechanism for reducing the impact of false positives would be to keep the amount of response small. If it actually is congestion, then this will result in several transitions between states B and B^1, before the congestion is relieved and the flow eventually makes the transition to state A. If on the other hand, this is a false positive, then due to the small amount of response, the flow does not lose much throughput.

This is the approach used in Vegas. Vegas uses additive decrease (by one packet) for early congestion response. However, note that this trades off the fairness properties of TCP in favor of maintaining high link utilization, since it has been shown in [69] that Additive Increase/Additive Decrease does not result in fair allocation between competing flows. In steady-state, if congestion avoidance is successful, then state C is eliminated and hence the flow spends most of its time transitioning between the

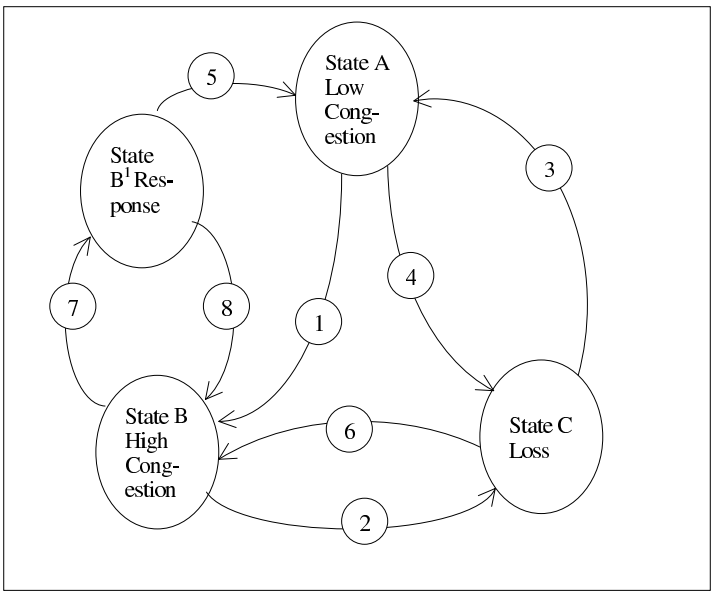


Fig. 40. State Diagram Showing the Transition between Different Congestion States in Flows with Early Response

states A and B - using AIAD for these transitions will result in compromising the fairness behavior of TCP.

Additionally, since the response is small, the buildup of the bottleneck queue will not be cleared out quickly. Hence, compared to a flow starting earlier, a flow that starts later may have a different idea of the minimum RTT on the path (in this case over-estimate it). This variable is used extensively in most end-host based schemes (including Vegas) to estimate what component of the RTT is due to the propagation delay and what component is due to queuing delay. Not clearing out the bottleneck queue completely, can hence result in offering an unfair advantage to a later starting flows, and hence result in getting more than its fair share of bandwidth.

An alternate mechanism for reducing the impact of false positives, is to respond probabilistically. When the probability of false positives is large, the probability that

a flow responds to the early congestion signal should be low and vice versa. Due to the probabilistic response, the protocol can retain the multiplicative decrease for early congestion response.

So, next we study the relationship between the false positives and the queue length at the bottleneck router. Fig. 41 shows the probability distribution of the normalized queue length when false positives are detected for the six cases of traffic loads discussed earlier, when the prediction signal is $srtt_{0.99}$. From the figure, we notice that false positives are more likely to occur when the queue length is smaller. Note that the traffic load for the six different cases consists of a mix of long term flows in both forward and reverse directions with different RTTs and also web traffic. For these cases of traffic load studied here, the false positives occur mostly when the queue length is less than 50% of the total queue size. While we may not be able to generalize the results for all possible types of traffic mix, it provides us an important insight: the uncertainties in congestion prediction are more likely to occur at lower queue lengths than at higher queue lengths, at least when smoothed RTT measurements are used as congestion predictors.

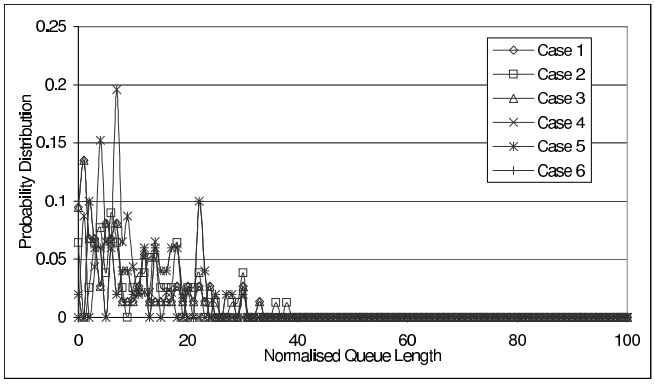


Fig. 41. Probability Distribution Function of Normalized Queue Length When False Positives Occur

Since the aim is to reduce the impact of false positives by designing an appropriate response function we argue that, when the queue size is small the response by a proactive scheme should be small and when the queue size is larger, the response should be larger. Note that this is conceptually similar to the probabilistic response function used in the AQM mechanism RED. Hence, for the response function, we emulate the probabilistic response function of RED. Due to the probabilistic nature of early response we call our scheme Probabilistic Early Response TCP (or PERT).

The probabilistic response of PERT is designed to be similar to that of "gentle" RED. Fig. 42 shows the probability of response against the congestion detection signal $srtt_{0.99}$. Similar to RED, we define two thresholds minthresh_ and maxthresh_ and the maximum probability of response maxP_. When the value of $srtt_{0.99}$ is below the minthresh_ the probability of response is 0. As the value of $srtt_{0.99}$ increases beyond minthresh_, the probability of reducing a window in response to each ACK linearly increases until it reaches the value maxP_ at maxthresh_. Between maxthresh_ and (2 * maxthresh_), the probability increases between maxP_ and 1. Beyond (2 * maxthresh_), the probability remains constant at 1 [2]. For the parameters minthresh_, maxthresh_ and maxP_ we use fixed values of (P+5ms, P+10ms and 0.05) respectively, where P is the propagation delay estimated by the minimum RTT observed by the flow. It is possible to choose these values adaptively based on network conditions similar to the mechanisms suggested in [91] - we are looking into this as part of our future work.

Our earlier results have shown the usefulness of making RTT measurements on every packet or ACK. Potentially, when queue lengths are observed to be above thresholds, every ACK arrival indicates congestion until the queue lengths fall below

[2]Different response functions can be chosen. "Gentle" RED is used here as a representative function.

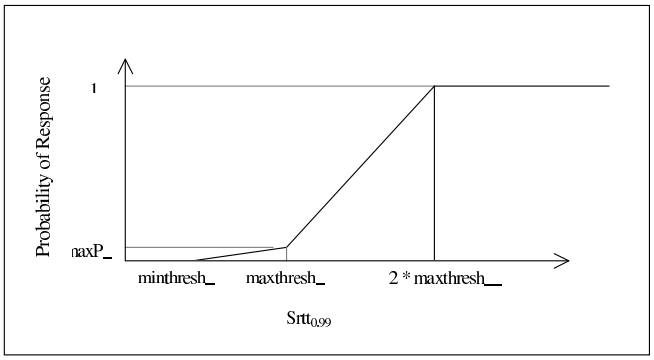


Fig. 42. Probabilistic Response Curve Used by PERT

the thresholds. It is not necessary for the flow to respond to each of these indications. The impact of response may not be seen until after an RTT. Hence, the early response to congestion is limited to once per RTT (even when random probability may pick multiple packets in one RTT).

For the early response, we use multiplicative decrease. When the proactive congestion response is successful, the queue lengths are expected to be maintained low. As a result, it is not necessary to respond with a 50% window reduction in case of early response. In [2] the authors show that the router buffers are set to the delay-bandwidth product of the link since the TCP flow reduces its window by 50%. If the TCP flow were to use a factor f instead for window reduction, then the relationship between the buffers and the window reduction factor can be re-written as $B > \frac{f}{1-f} * BDP$. Based on this, for a conservative value that the queue length doesn't exceed half its capacity and that the capacity of the buffer is set to one BDP according to the rule of thumb, we choose the window decrease factor to be 35%. When a flow responds early due to the congestion perdition signal, it decreases its congestion

window by $0.35 * W$ [3].

Note that the choice of the amount of response is a trade off. Since the flows respond before the bottleneck queue is full, a large multiplicative decrease can result in lower link utilization. On the other hand, decreasing the amount of response could result in the bottleneck link buffers not getting entirely cleared, resulting in unfairness among flows starting at different times, as discussed earlier.

While it has been a natural choice based on our observations to emulate the probabilistic marking of RED, it is possible to replace the probabilistic response curve with the algorithms used in other AQM schemes as well. In the next section, we present extensive evaluation of PERT that emulates RED/ECN. Following this evaluation, we present some preliminary results of the emulation of an alternate AQM mechanism, REM[74]. As part of our future work in this area, we will continue to investigate the possibility of emulating other AQM mechanisms as well.

C. Experimental Evaluation

We have conducted extensive ns-2 based simulations to evaluate PERT. Difficulties in simulating the Internet have been discussed in [92]. We attempt to make our evaluation realistic by simulating a wide range of network parameters. We first present the results for a single bottleneck topology with bottleneck link bandwidth in the range of [1Mbps, 1Gbps], RTT in the range of [10ms, 1s], the number of long-term background flows in the range of [1, 1000] and the number of web sessions in the range of [10, 1000]. We then evaluate the impact of multiple bottleneck links and flows of different RTTs. All simulations are run for 400 seconds and reported results are measured during the stable period between 100 and 300 seconds to show

[3] If a packet loss occurs, then the response will be similar to that of standard TCP variants and the fast retransmit/recovery algorithms are triggered.

the steady state behavior. Next we evaluate the dynamic behavior due to transient changes in traffic load where sudden changes in available bandwidth are caused due to the arrival and departure of flows. For all experiments, the bottleneck buffer size is set to the bandwidth-delay product with the minimum number of packets being equal to at least twice the number of flows. When multiple flows share a link, their start times are chosen randomly in the range (0,50) seconds to illustrate the impact of flows starting at different times on the fairness as discussed in the previous section. We present all results in comparison to (a) Sack, when Droptail routers are used, denoted as Sack/Droptail (b) ECN-enabled Sack, when RED routers are used, denoted as Sack/RED-ECN and (c) TCP-Vegas, denoted as Vegas. For experiments with TCP-Vegas and PERT, the bottleneck link routers use the default Droptail buffer management.

1. Impact of Bottleneck Link Bandwidth

In this experiment, the bottleneck link bandwidth is varied from 1Mbps to 1Gps. The end-to-end RTT of the flows is set to 60ms and the number of flows is varied such that the link is efficiently utilized even at large bandwidth. Fig. 43 shows the average bottleneck link queue length, the bottleneck link drop rate, the link utilization and the Jain Fairness Index [69] of the competing flows. From the graph, we see that PERT's average queue length is similar (and in some cases better) than Sack/RED-ECN. As expected, the average queue length with Sack/Droptail remains high across most experiments. A surprising result is that TCP-Vegas has higher average queue length than Sack/Droptail in some cases. Higher drop rates observed with Sack/Droptail relieve the congestion while Vegas's maintenance of 1-3 packets in the buffer (based on the α and β parameters) could lead to high queue lengths, while avoiding packet losses.

All the proactive mechanisms (Sack/RED-ECN, PERT and TCP-Vegas) maintain zero losses in most cases. The link utilization of PERT is lower than TCP-Sack in the cases where the bottleneck link bandwidth is small resulting in short buffers, but in the other cases, it is similar. TCP-Vegas maintain high link utilization, but it comes at the cost of fairness among competing flows. The fairness among PERT flows is similar to that of standard TCP, with the Jain Fairness Index being close to 1.

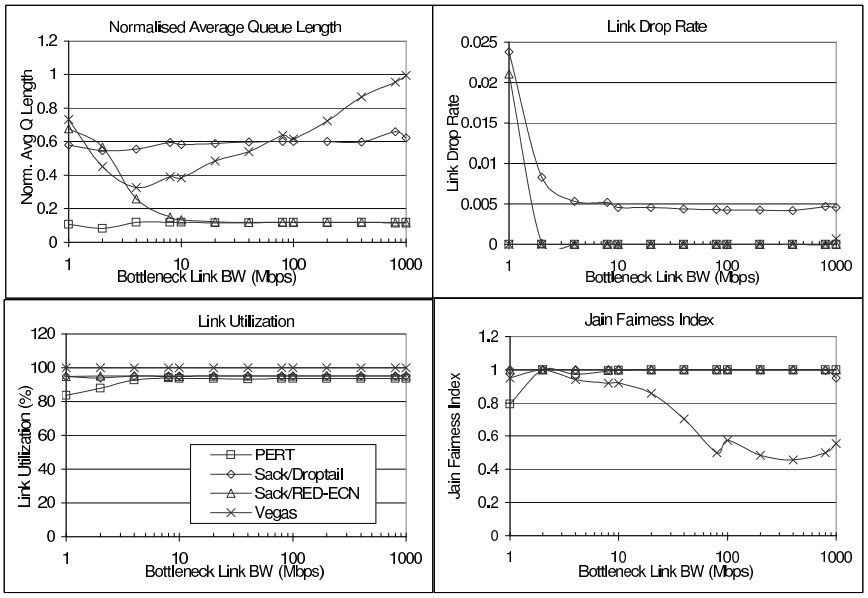


Fig. 43. Impact of Bottleneck Link Bandwidth (Note: X-axis is in Log-scale)

2. Impact of Round Trip Delays

In this experiment, the bottleneck link bandwidth is 150Mbps and the number of flows is 50. The end-to-end delay is varied in the range of 10ms to 1 second. Fig. 44 shows the results. From the figure, we see that the average bottleneck link queue length and the drop rate are similar in case of PERT and SACK/RED-ECN. Sack/RED-ECN

itself has lower link utilization, but it is slightly better than that of PERT, since we have used the adaptive RED version for the routers that tunes the parameters according to the network conditions and PERT uses fixed thresholds. The Jain Fairness Index remains high indicating that the throughput is shared in a fair manner among the 50 flows.

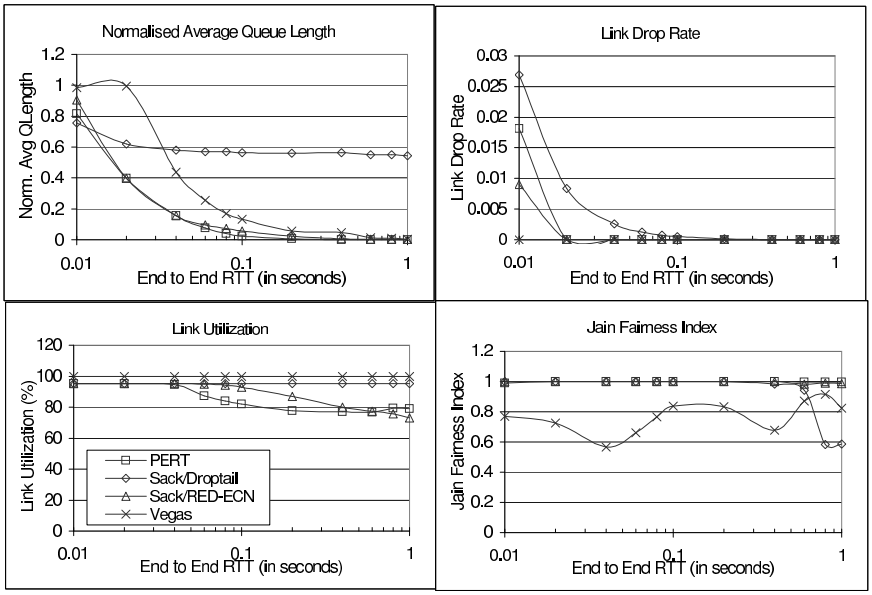


Fig. 44. Impact of End-to-End RTT (Note: X-axis is in Log-scale)

3. Impact of Varying Number of Long-term Flows

In this experiment, the bottleneck link bandwidth is set to 500Mbps and the number of long-term flows is varied from 1 to 1000. The end-to-end delay is 60ms. Fig. 45 shows the results. Again the average bottleneck link queue length and the bottleneck link drop rate of PERT are similar to that of SACK/RED-ECN. The Jain Index remains high even when the number of flows is large. Link utilization of Sack/RED-ECN is slightly higher than PERT. Vegas tries to maintain a fixed number of packets

in the queue and as a result, as the number of flows increases, so does the average queue length and the drop rates. Vegas's link utilization remains high, while the Jain Fairness Index remains low.

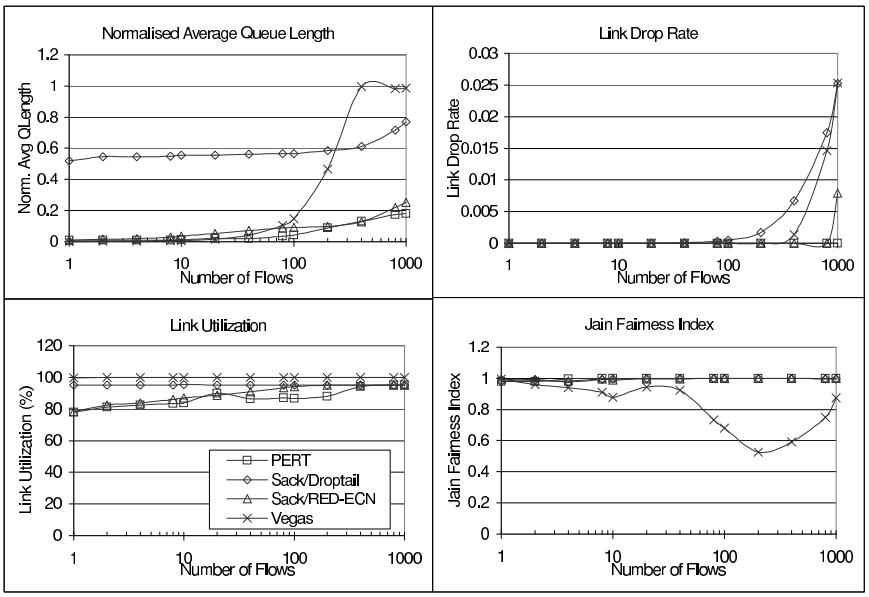


Fig. 45. Impact of Varying the Number of Long-term Flows (Note: X-axis is in Log-scale)

4. Impact of Web Traffic

We now consider simulations with web traffic to understand the impact of bursty traffic. The bottleneck link bandwidth is set to 150Mbps, the end-to-end delay is 60ms and the number of long term flows is 50. The number of web sessions that share the bottleneck link with these long-term flows is increased from 10 to 1000. For the web traffic, the parameters are chosen based on the guidelines in [93]. As seen from Fig. 46, as the load offered by the web traffic increases, the average link queue length remains low and as a result no packet losses are observed in case of PERT,

similar to that of Sack/RED-ECN. The link utilization of PERT is slightly lower than that of SACK/RED-ECN. The Jain Fairness Index of the long-term flows remains high.

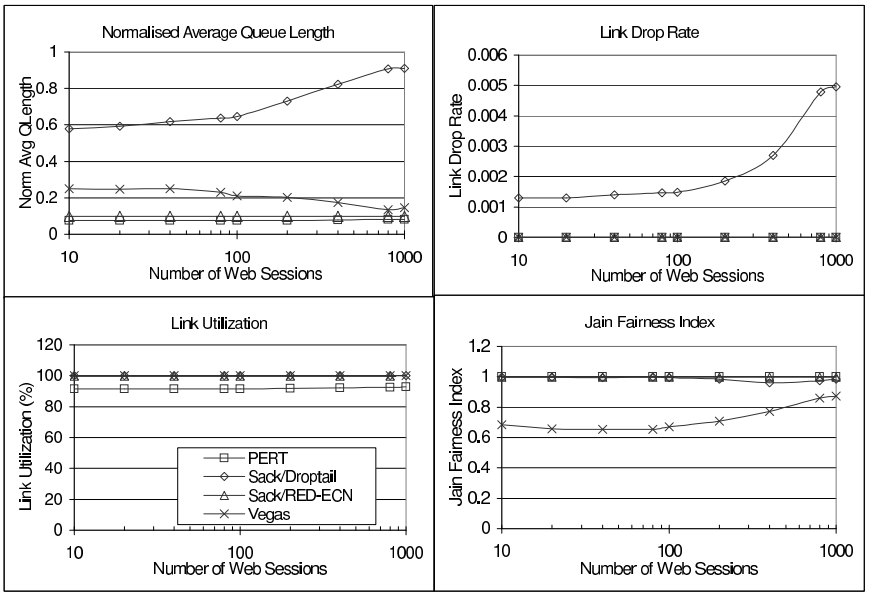
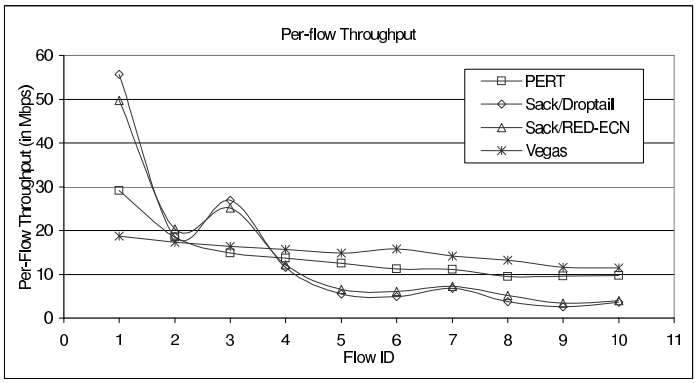


Fig. 46. Impact of Varying the Number of Web Session (Note: X-axis is in Log-scale)

5. Impact of Different RTTs

In the following experiment, a bottleneck link with 150Mbps capacity is shared by 10 flows with end-to-end delays being 12ms, 24ms, 36ms and so on, to 120ms. Hence the ratio of the RTTs of the smallest RTT flow to the largest RTT flow is 10. 100 web sessions are run in the background. Fig. 47 shows the throughput achieved by each flow. From the graph we see that PERT and Vegas may reduce the RTT unfairness inherent to TCP. From the table below the graph, we see that the link utilization of PERT is similar to SACK with either Droptail or RED/ECN, while the average queue length and drop rate are lower. Because of the improved sharing of bottleneck

link bandwidth, the Jain Fairness Index also shows an improvement.



	PERT	Sack/Droptail	Sack/RED-ECN	Vegas
Link Utilization	93.81	93.77	93.90	99.99
Norm Avg Qlength	0.28	0.42	0.41	0.07
Link Droprate	3.98E-06	7.18E-04	4.95E-04	0
Jain Faimess Index	0.86	0.44	0.51	0.98

Fig. 47. RTT Unfairness

6. Multiple Bottlenecks

The congestion signal in PERT measures the end-to-end RTT, and hence conveys information about the combined queue lengths along the path. Router based AQM schemes, on the other hand, estimate the queue length locally, then use ECN to convey the information end-to-end. In this experiment we investigate the impact of multiple bottleneck links. The topology, shown in Fig. 48, consists of six routers labeled R1 to R6. The links between routers have a capacity of 150Mbps and a delay of 5ms. Each router is connected to a cloud of 20 nodes with a link of capacity 1Gbps and delay 5ms. The nodes in each cloud send data to the nodes in the cloud connected to the adjacent router. Also, all the nodes in the cloud connected to router R1 also send data to the nodes in the cloud connected to R6.

Fig. 49 shows the average queue length, drop rate and utilization of the link

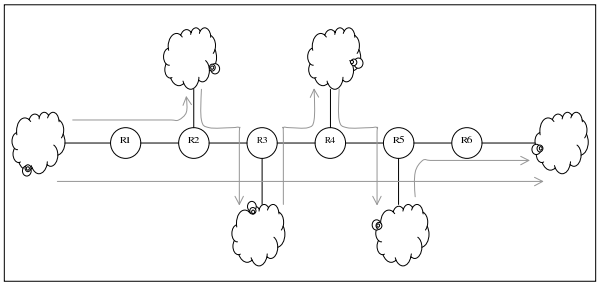


Fig. 48. Topology Used for Understanding the Impact of Multiple Bottleneck Links

between each pair of routers as well as the Jain fairness Index of all the flows between each pair of routers. From the figure we see that PERT maintains low queue length and zero drop rates across all the bottleneck link queues. The link utilization is similar to that of Sack/RED-ECN and the fairness among flows passing through the common set of routers is maintained.

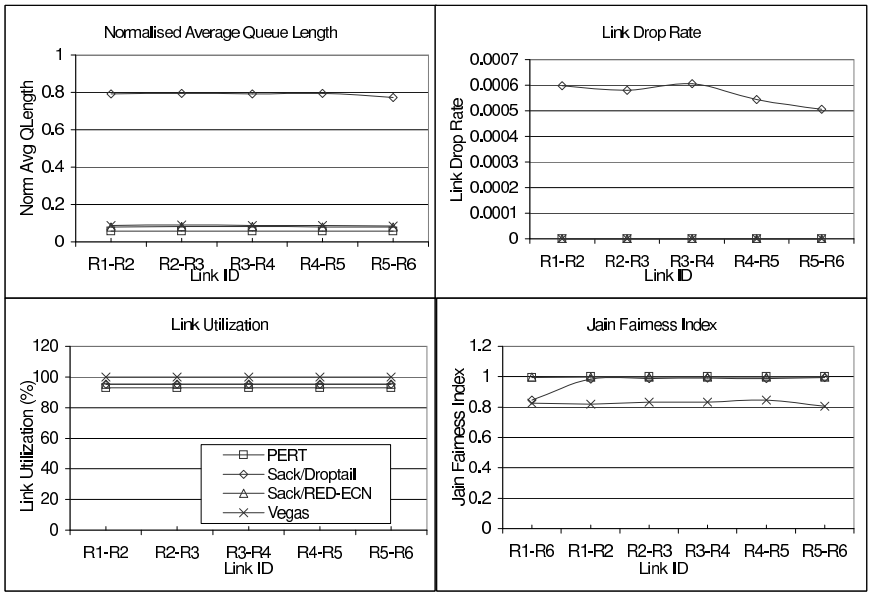


Fig. 49. Impact of Multiple Bottleneck Links

7. Dynamic Protocol Behavior

In this experiment, we study the dynamic protocol behavior of the different schemes. Unlike previous experiments which focused on steady state behavior, the results of the first few seconds of simulation time are not discarded to illustrate the impact of transients. These experiments investigate the responsiveness of the different schemes to sudden changes in traffic.

In this experiment, 25 PERT flows are started at time 0 seconds. Starting at 100 seconds, for the next 300 seconds 25 new flows are added at 100 second intervals, causing severe contention for available bandwidth. Starting at 400 seconds, 25 flows leave the network at 100 second intervals creating a sudden availability of bandwidth. We repeat the experiment with Sack/Droptail, Sack/RED-ECN and Vegas. Fig. 50 shows the the aggregate throughput of the set of flows that start together. From the figure it is clear that the PERT flows respond quickly to dynamic changes in network bandwidth, similar to Sack/Droptail and Sack/RED-ECN. Vegas exhibits previously observed unfairness among competing flows.

In the above experiment, the sudden changes were created by the joining and leaving of flows that are responsive to congestion indications. In the next experiment we used UDP source to create a "square wave" traffic load, that is not responsive to congestion. The UDP source was started at 50 seconds with the sending rate set to half the link capacity (75Mbps). It stayed "on" for 50 seconds and was "off" for the next 50 seconds. This was repeated constantly to generate the "square wave" traffic pattern. Fig. 51 shows the aggregate throughput of the PERT flows and the UDP flow. From the figure, we see that when the available bandwidth is suddenly reduced by half, PERT flows quickly stabilize to the new operating point and when the bandwidth becomes suddenly available PERT flows can quickly use it up. Similar

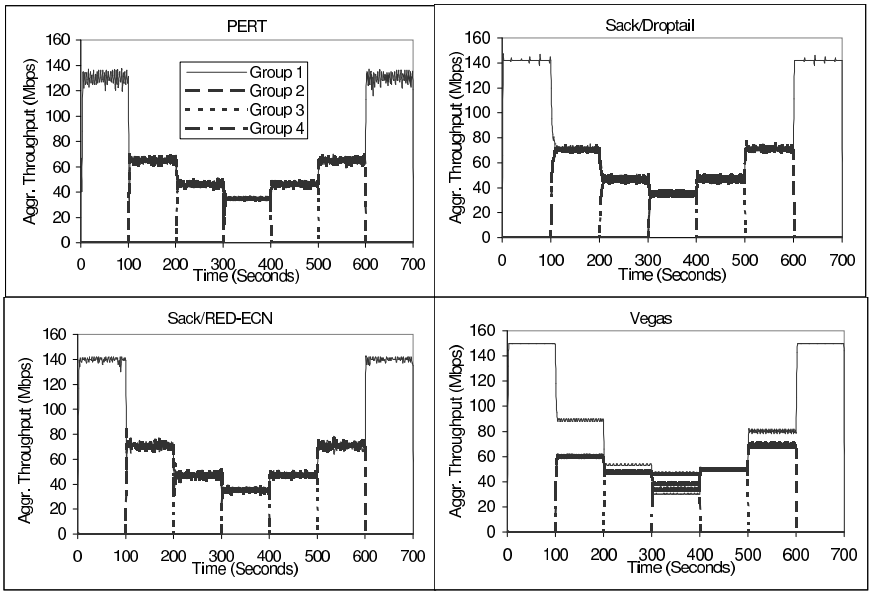


Fig. 50. Response to Sudden Changes in Responsive Traffic

results were observed with Sack/Droptail, Sack/RED-ECN and Vegas and so have not been included here. For this simulation, the link utilization over the entire period of the simulation was 91.76%, the average queue length was 9.69% and the drop rate was 0.

8. Emulating Other AQM Algorithms

In this section, we present the preliminary results of emulating REM [74] with packet marking. In REM two different components are used for determining the 'congestion price', based on which, the marking probability is determined. The two components that determine the congestion price are (a) mismatch in the incoming rate and the service rate of the link and (b) mismatch in the queue length compared to some predefined threshold. Note that the second component is similar to that used in RED. The first component on the other hand ensures that as the number of flows increases

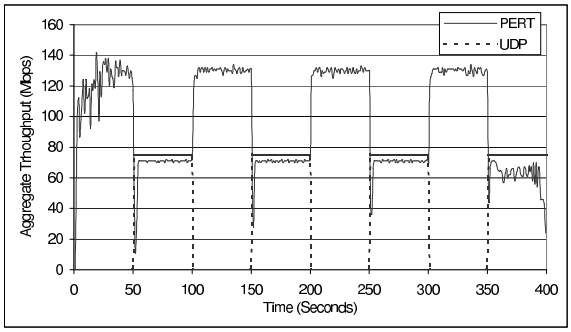


Fig. 51. Response to Sudden Changes in Non-responsive Traffic

(i.e., the arrival rate at the link increases), so does the price and consequently, the marking probability. This ensures that, at larger number of flows, large queue length is not required for higher marking probability.

Specifically, the equations used by REM for computing the congestion price and marking probability are

$$p_l(t+1) = [p_l(t) + \gamma(\alpha_l(b_l(t) - b_l^*) + x_l(t) - c_l(t))]^+ \tag{4.1}$$

$$m_l(t) = 1 - \phi^{-p_l(t)} \tag{4.2}$$

where $\gamma > 0$, $\phi > 0$ and $\alpha_l > 0$ are small constants, $[z]^+ = \max\{z, 0\}$, $b_l(t)$ is the aggregate buffer occupancy of queue at link l in the period t, $b_l^* \geq 0$ is the target queue length, $x_l(t)$ is the aggregate input rate at the queue at link l in period t, $c_l(t)$ is the available bandwidth at the queue at link l in period t and $m_l(t)$ is the marking probability at the queue at link l in period t.

We can detect the queue mismatch from the end-host similar to before, by comparing the delay with a predefined threshold. In order to detect rate mismatch, consider the following: when the arrival rate at a link is larger than the link capacity, the extra packets will be saved in the bottleneck link buffer. In this case, the rate

mismatch can be detected by measuring the change in buffer length or in the case of end-hosts, by monitoring the change in delay. However, when the arrival rate at a link is less than the link capacity, and the buffers are empty, the rate mismatch is negative. End-hosts will not be able to measure this since the buffers are empty and RTT will be equal to the propagation delay on the link. This negative rate mismatch will help decrease the price in case of REM and hence the marking probability. We emulate this by allowing the price to decay, when the price is non-zero and remains unchanged over subsequent measurements.

We use a logarithmic decay function for reducing the price. For each packet that yields a zero change in price, when the price is positive, a counter *ctr* is incremented and the price is decayed using $price = price * 0.999^{ctr}$. This form of function ensures that when the value of *ctr* is small, the change in price is negligible, but when the value of *ctr* increases, the price decays rapidly. Note that the choice of this function is a trade-off. If the price decays too rapidly, then it gives the queues at the bottleneck link a chance to build up and as a result the average queue length will be higher. On the other hand, if the price decays too slowly, then the flow continues to have window reductions even when the arrival rate at the bottleneck link is lower than the service rate, resulting in reduced link utilization. Preliminary results presented here show that the logarithmic decay function performs relatively well, under a wide range of network conditions. Under some network condition, this may however result in the end-host based solution having slightly different behavior than router-based REM.

Router-based REM uses several different parameters. We tuned the parameters of the end-host based solution, using the experiment similar to that in Section IV.C.3, to obtain performance similar to router-based scheme. In this experiment, the impact of varying the number of flows sharing the bottleneck link is studied. Fig. 52 shows the results. The curve marked SACK/REM-ECN shows the performance of Sack

flows when the bottleneck link router uses REM with ECN marking. The curve marked PERT-REM shows the end-host emulation. For the end-host emulation the REM parameters in Eq.4.1, (γ, α and ϕ) were chosen to be (0.02, 0.8 and 1.001). The target queue length is measured using *target_delay*, which was chosen to be twice the resolution of the RTT measurement at the sender. The reason for choosing *traget_delay* based on resolution of RTT is that, unlike the emulation of RED where the queue mismatch determines the probability directly, the emulation of REM uses the queue mismatch for determining the price. If the *target_delay* is chosen to be less than the resolution of RTT changes, then the error accumulates in the value of of price, resulting in continuous increase in the response probability and eventually, unnecessary proactive window reductions. The resolution in RTT is computed by measuring the most likely value of the positive changes in the observed round trip time. Similar to the emulation of RED/ECN discussed in the previous section, the window reduction for early response was chosen to be 0.35.

As seen from the graph, the average bottleneck link queue length, the bottleneck link drop rate, the link utilization and the fairness among competing flows is similar with the end-host based emulation of REM to that of using Sack with REM routers using ECN marking.

Next we investigate the impact of the bottleneck link bandwidth. This experiment is similar to that in Section IV.C.1. Fig. 53 shows the results. From the figure we see that, at higher bottleneck link bandwidth the performance of the end-host emulation is similar to using REM at the routers. However, at lower bottleneck bandwidth, the end-host emulation maintains lower average queue length and lower bottleneck link droprate than the router-based scheme. This comes at the cost of slightly lower link utilization at lower bandwidth. Also, at lower bandwidth, the delay for transmitting each packet is high. This results in very coarse resolution of

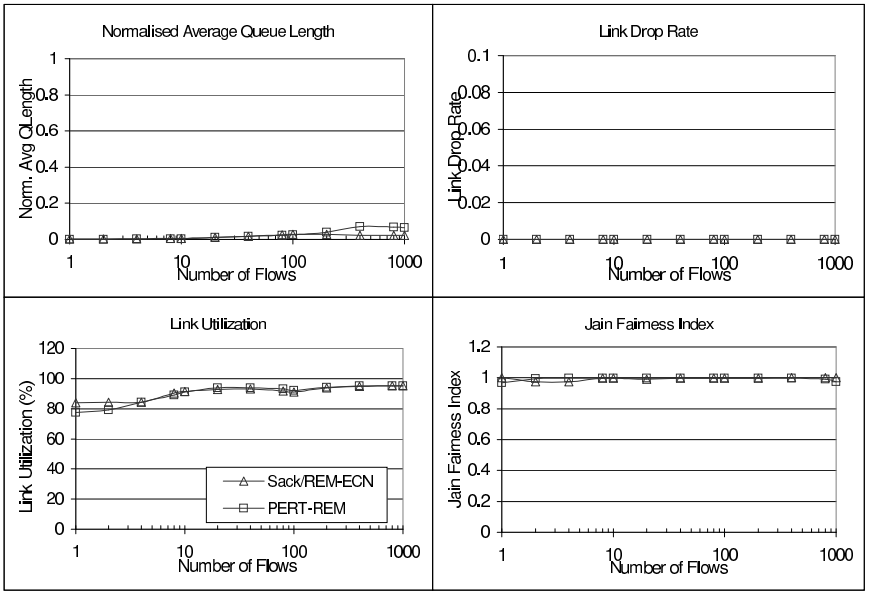


Fig. 52. Preliminary Results of Emulating REM at End-hosts as the Number of Flows is Varied (Note: X-axis is in Log-scale)

the RTT measured at the end-host. Because *target_delay* is chosen based on this resolution, the link utilization remains above 80%, but the fairness among competing flows at is slightly reduced.

Next we study the impact of the end-to-end propagation delay. This experiment is similar to that in Section IV.C.2. Fig. 54 shows the results. From the figure we see that, the link utilization of end-host emulation remains similar to that of the router-based scheme. When the RTT is low, the average link queue length, the drop rate and the fairness of the end-host emulation is slightly worse than the router-based scheme. At higher RTTs the performance is similar.

The results presented here illustrate that it is possible to emulate other router based schemes in the end-host response function of PERT, so that sender-based schemes may maintain low bottleneck link queue length and droprates, without re-

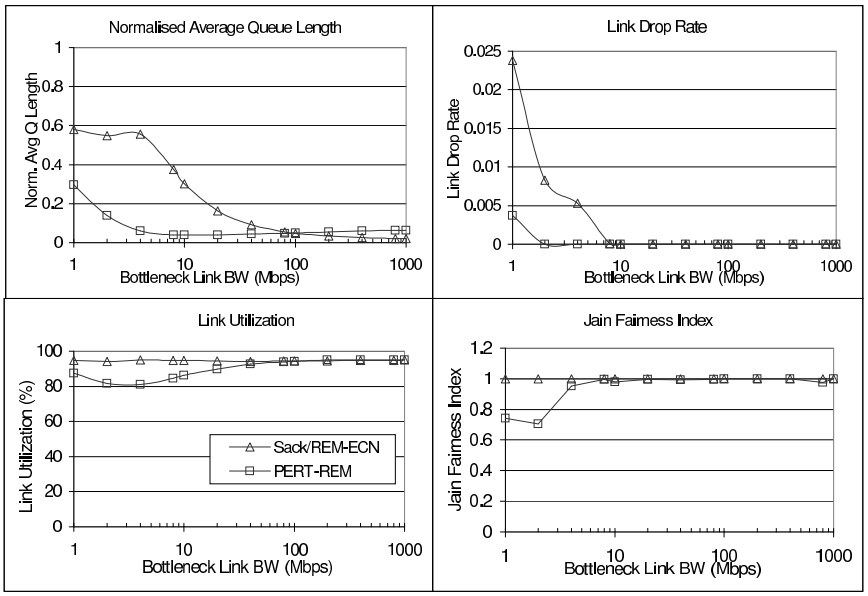


Fig. 53. Preliminary Results of Emulating REM at End-hosts as the Bottleneck Link Bandwidth is Varied (Note: X-axis is in Log-scale)

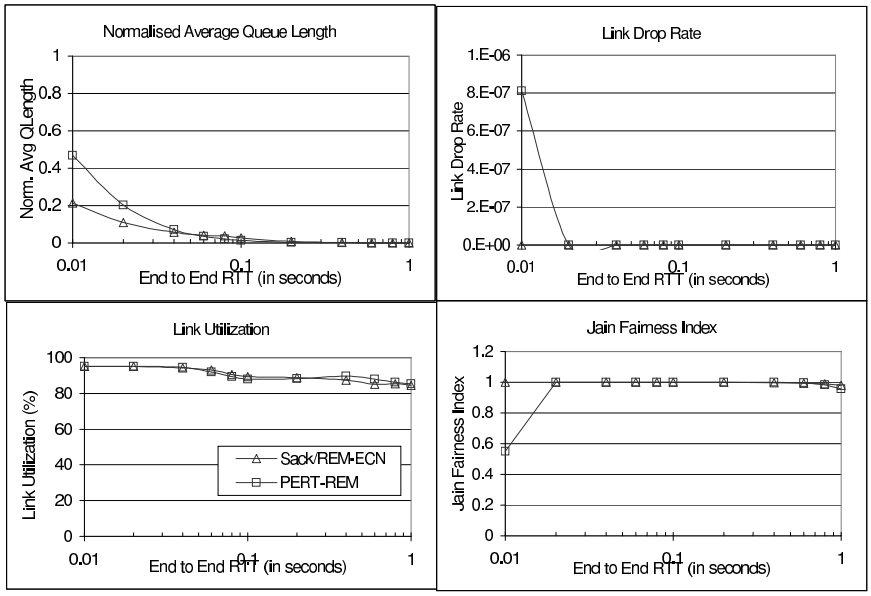


Fig. 54. Preliminary Results of Emulating REM at End-hosts as the End-to-end RTT is Varied (Note: X-axis is in Log-scale)

quiring any router support. The work here is preliminary and more study is required.

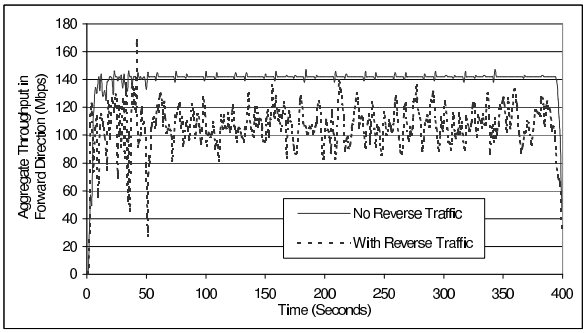
D. Discussion

While the results presented in Section C are highly encouraging we have identified some open issues, which require further study. Here, we discuss the open issues and some of the possible solutions for addressing them.

Impact of Reverse Traffic: Similar to earlier schemes using delay-based congestion avoidance, we use the round trip time to predict the build-up of bottleneck queues. However, since round trip time is the sum of delays in both the forward and reverse direction, congestion in the reverse path can trigger an early response. Whether congestion response should be only for forward path congestion, like the current versions of TCP or for congestion in both directions is debatable. For instance, when the reverse path is severely congested TCP-SACK sees much lower utilization due to loss of ACKs resulting in timeouts.

Fig. 55 shows the link utilization when 50 SACK flows share a bottleneck link of capacity of 150Mbps when (a) there is no reverse traffic and (b) when the reverse direction has 50 SACK flows as well. From the figure we see a significant drop in link utilization of the flows in the forward direction (95% to 72%), when reverse path is also congested. The fraction of timeouts has significantly increased (two orders of magnitude) for the simulation with reverse traffic compared to the one without. Since timeouts are significantly more expensive than triggering congestion avoidance, responding to reverse path congestion may be justified in this case. In this study, we do not address this issue. It must be noted however, that if responding to reverse path congestion is not acceptable, then PERT can be used with one-way delays to achieve similar benefits. Methods for computing one-way delays have been studied in

[88, 94].



Statics For Traffic in Forward Direction			
	Link Utilization (%)	Link DropRate	Total Timeouts / Total number of Packets Transmitted
No Reverse Traffic	95.24	1.23E-03	1.15E-06
With Reverse Traffic	71.79	6.26E-03	4.11E-04

Fig. 55. Impact of Reverse Traffic on SACK Flows

Co-existence with Non-proactive Flows: Another issue with end-host solutions is that of co-existence with non-proactive flows. When several flows share a bottleneck link, if some of them proactively respond to congestion, while others don't, then the flows that respond early will see degraded performance. This problem can be addressed relatively easily in router-based schemes, by dropping the packets instead of marking them when the queue length exceeds a certain threshold (like RED does when queue length exceeds 2*maxthresh_ in case of "gentle" version, maxthresh_ otherwise). This forces the non-proactive schemes to also back off before the bottleneck link queue fills up. Note that, the tradeoff here is a slightly increased packet drop rate, to maintain low queuing delay. With an end-host based scheme, each flow only has control over its own congestion window and hence emulating something similar is not possible. However, it is possible for a flow to reduce its pro-activeness, when it notices that the queue length continues to increase in spite of its repeated early window reductions.

We are currently investigating mechanisms for making the pro-activeness adaptive based on the flow's perception of how effective early response has been. A number of possibilities exist: increasing the time for next response progressively if queue lengths persist, making the probability of response a function of the time since the last response, limiting the probabilistic early response to once when the probability exceeds some threshold (say 0.75) etc. Alternately, as suggested in Sync-TCP[88], the increase function can be made more aggressive than TCP in the absence of congestion to compensate for the loss in throughput in the presence of congestion.

Emulating Other AQM Algorithms: The preliminary results for the emulation of REM brings up some interesting issues. First, when the rate mismatch is positive, it can be predicted from the sender by observing the change in rtt, and the sending rate can be reduced appropriately. However, when the rate mismatch is negative, a case where the flow should continue to increase its sending rate, a sender has no means of measuring the rate mismatch. In this study we have used a logarithmic decay function to ensure that the flow does not continue to respond proactively when this happens. But, future work in this area will need to investigate how to emulate this to reflect the actual changes at the bottleneck link. Rate mismatch is used in several different router-based schemes, and solving this problem will allow several different rotuer-based schemes to be emulated more accurately, from the end-hosts.

Secondly, the resolution of the changes in RTT observed at end-hosts imposes some limitations on the performance of the end-host emulation. When the bandwidth is low, the packet delay will be large, resulting in coarse resolution of the changes in end-to-end RTT. When the end-to-end delay is low, the resolution of changes in RTT may be large, relative to the end-to-end delay. In these cases, the performance of end-host emulation may not be as good as using the actual router-based scheme, which can have better resolution by monitoring the queue size or the rate mismatch

in packets.

Other Key Differences between Router-based AQM and End-host Emulation: In case of router-based AQM, the centralized bottleneck link router determines the marking probability and the signal is sent only to the flow(s) that needs to respond. In the end-host based emulation, each flow estimates the network congestion independently and at any point, potentially all the flows have the congestion signal. Hence, even though the long term average probability of response is similar for the two approaches, distributions of probabilities at individual flows could be different. These differences may lead to slightly different choice of parameters at end-hosts than at the routers. We are currently looking at mechanisms for translating the centralized probability distribution used in a router-based scheme to the distributed probability distribution used by the end-host to generalize the approach.

In case of router-based AQM, the end-hosts receive congestion information only when the marking algorithm used by the AQM scheme marks the packets of its flow. With end-host AQM, potentially all the end-hosts will be able to identify the onset of congestion at the same time. This could lead to more options in designing the response function.

Some of the algorithms used in the AQM schemes need an estimation of the round tip time of the flow to determine the feedback [61, 79] to the end-host. The routers use an average value for this purpose. With an end-host based emulation, the flow knows its RTT, providing options for different new designs.

End-host based emulation uses end-to-end delay as a congestion measure and hence can ensure that delays have a tighter bound end-to-end, compared to the router-based AQM where each router can offer bounds only locally.

E. Conclusions

In this chapter, we showed that congestion prediction at end hosts is more accurate than characterized by previous studies based on flow level measurements. We show that while it is necessary to improve further the accuracy of end-host delay-based congestion predictors, the impact of any inaccuracies can be mitigated by the choice of an appropriate response function. We have presented here a scheme called PERT, which emulates RED/ECN like behavior in the response function. PERT is shown to offer benefits similar to using RED at the routers with ECN marking, but without the router support. Our results, based on a wide array of network conditions, indicate that PERT can efficiently maintain low queue lengths and almost zero losses, while retaining high degree of fairness. The link utilization is slightly lower, but in most cases, similar to that of ECN-enabled SACK in the presence of RED routers with packet marking. The proposed scheme is flexible in that, other AQM schemes can be potentially emulated at the end-host. We illustrate this by presenting the preliminary results for the emulation of REM/ECN in the response function of PERT.

CHAPTER V

CONCLUSION: PUTTING IT ALL TOGETHER

In this chapter, we investigate how the different proposed modifications to TCP interact with each other. We first investigate the impact of packet reordering in high-speed networks and show that using delayed congestion response proposed in Chapter II with LTCP proposed in Chapter III can significantly improve TCP performance in high-speed networks with packet reordering. We refer to this new flavor as LTCP-DCR. Next, we investigate the impact of high-speed protocols on the bottleneck link buffers and the packet losses occurring at the bottleneck link. We show that when LTCP is combined with PERT proposed in Chapter IV, (referred to as LTCP-PERT), the packet losses at the bottleneck router can be eliminated and the average queue length at the router can be maintained low. Finally we combine LTCP-DCR with PERT (the new flavor is referred as TCP-LDP) to obtain the benefits of delay-based congestion avoidance of PERT, while maintaining high link utilization even on high BDP links with high levels of packet reordering.

A. LTCP-DCR : Dealing with Packet Reordering in High-speed Networks

In Section II.B.2, we have shown that packet reordering can result in significant degradation of TCP throughput. This problem is magnified in high-speed networks with low levels of multiplexing, where the window size needs to grow to large value to be able to fully utilize the link. Responding to packet reordering events as if they were packet losses, causes the window to remain small, thus resulting in drastic reduction in link utilization. In this section, we investigate the use of delayed congestion response with LTCP to fully utilize a high BDP link in the presence of packet reordering.

1. Impact of Packet Reordering

Fig. 56 shows the throughput of the different flavors of TCP as the percentage of packets reordered is increased. The simulation consists of one flow over a bottleneck link of capacity 1Gbps and end-to-end RTT of 50ms. Packet reordering is simulated by randomly choosing packets based on a uniform distribution and delaying them. The packet delay is chosen from a normal distribution with mean 25ms and variance 8ms. The high-speed TCP protocol flavors such as LTCP, BIC and HTCP suffer dramatic degradation in performance with the throughput remaining similar to that of TCP-SACK, since the packets that are delayed enough to cause three dupacks will result in window reduction. This is in agreement with the response curve in Fig. 28 showing that at large loss rates, the behavior of the high-speed protocols is similar to that of TCP-SACK. Since there is no way to distinguish packet reordering from packet losses, the behavior of high-speed protocols is similar to that under high packet loss rates.

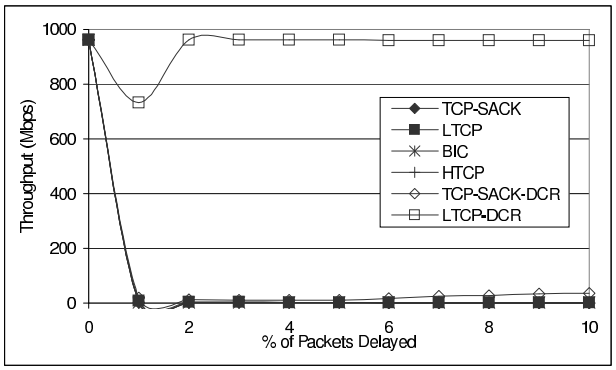


Fig. 56. Throughput in the Presence of Packet Reordering in High-speed Networks

We refer to the TCP-SACK flavor enhanced with DCR as TCP-SACK-DCR. With TCP-SACK-DCR, the performance is slightly improved. However, since the

packet delay is chosen randomly from a normal distribution, a small percentage of packets may be delayed by more than one RTT. Since the delay in response is chosen to be one RTT, these packets will trigger window reduction. TCP-SACK uses additive increase of one packet per round trip time, and hence takes long time to recover from window reduction resulting in low link utilization. Fig. 57 shows the number of times fast retransmit algorithm is triggered as a fraction of total packets sent. It can be seen that TCP-SACK-DCR reduces the number of times fast retransmit is triggered compared to TCP-SACK. However, since it cannot increase the window size quickly after a window reduction, the fraction of false retransmits compared to the total packets is still relatively high and hence the throughput achieved remains low.

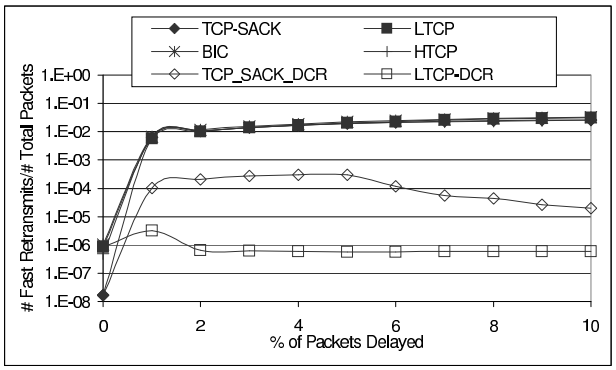


Fig. 57. Fraction of Packets That Trigger the Fast Retransmit/Recovery Algorithms

On the other hand, when a high-speed protocol such as LTCP is enhanced with DCR (LTCP-DCR), due to the combined ability of DCR to avoid responding to most packet reordering events and the ability of LTCP to quickly increase the window after a false response, the throughput is maintained high. Note that, for the same percentage of packets delayed, the fraction of false retransmits to total packets is much lower in case of LTCP-DCR compared to TCP-SACK-DCR due to the aggressive increase algorithm employed.

Note that, in order to successfully achieve high link utilization, it is important to have both the components - delayed congestion response and quick recovery from window reduction for packets delayed by more than one RTT. It is possible that packets are delayed by more than one RTT during the beginning of a flow, in which case the congestion control algorithm may not have a chance to reach an aggressive mode. In such a case, the achieved link utilization may be lower than optimal. One such case is shown in our results when the percentage of packets delayed is 1%. Fig. 58 shows the evolution of window when 1% and 2% of packets are delayed. Due to the random nature of the delays, in case of 1% packets getting delayed, several packets are reordered by unrecoverable delays during the early stages of the flow's window evolution. This slows how quickly the layers are increased in LTCP and hence it operates at suboptimal layers. As a result, the link utilization is suboptimal. Note however, that in spite of this, the throughput achieved by LTCP-DCR is about 2 orders of magnitude more than LTCP, when 1% of packets are delayed.

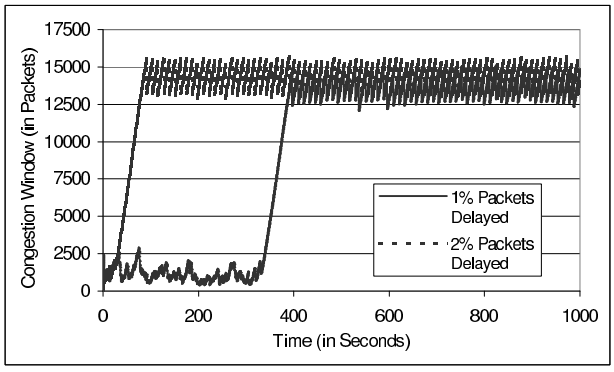


Fig. 58. Window Evolution of LTCP-DCR under Different Conditions of Packet Delays

2. Behavior of LTCP-DCR in the Absence of Packet Reordering

In order to ensure that the use of delayed response with LTCP does not alter the fairness characteristics in the absence of packet reordering, we conducted the following simulations. In the first set of simulations the total number of flows in the network was increased from 10 to 100, resulting in congestion loss rates ranging from 0.03% to 0.6%. In each case, half of the flows use LTCP and the other half used LTCP-DCR. Fig. 59 shows that the fairness index remains high in most cases, indicating that using delayed congestion response with LTCP does not make it unfriendly to LTCP flows that do not use delayed response.

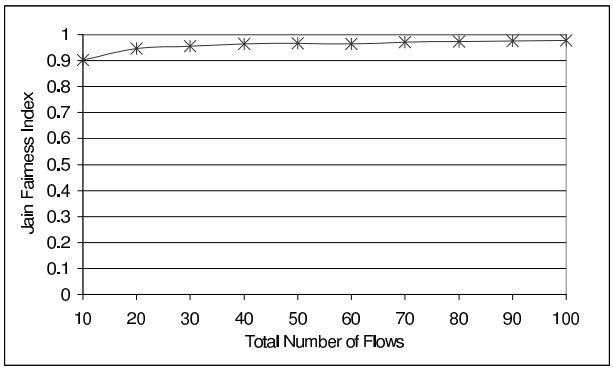


Fig. 59. Fairness between LTCP-DCR and LTCP Flows

Next we study the impact of delaying congestion response on bottleneck link drop-rate. First we run 50 flows of LTCP with the bottleneck link buffer-size varied from 0.1BDP to 2BDP and note down the bottleneck link drop-rate. Next we repeat the experiment with 50 LTCP-DCR flows. Fig. 60 shows the results. From the graph we see that, the drop-rate is slightly higher in case of LTCP-DCR compared to LTCP. However, the difference is not significantly high and the link utilization remains similar in both the cases (96.1% when larger buffers are available and 95.5% at lower

buffers).

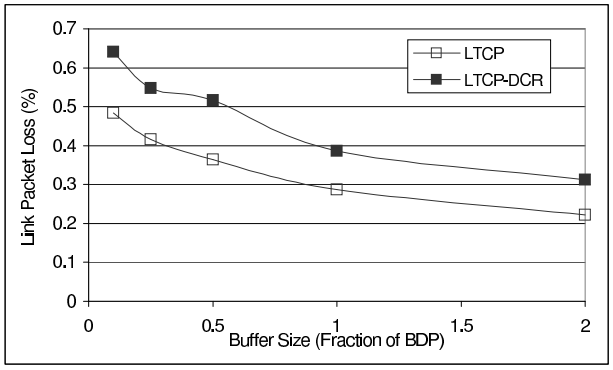


Fig. 60. Comparison of Drop-rates for LTCP-DCR and LTCP Flows at Different Buffer Sizes

3. Behavior of LTCP-DCR in the Presence of Both Packet Reordering and Congestion

Finally, we investigate the behavior when both packet reordering and congestion is present in the network. This experiment is similar to the earlier experiment with only packet reordering - however, the bottleneck link is now shared by 50 flows. Fig. 61 shows the results. From the figure we see that the aggregate link utilization by LTCP, BIC and HTCP is slightly improved compared to Fig. 56 because of using 50 flows. However, when packet reordering in non-zero, the aggregate link utilization is still drastically lower than when there is no packet reordering. In case of LTCP-DCR, high link utilization is maintained even when 10% of the packets are delayed.

The sum of the number of times fast retransmit is triggered as a fraction of total number of packets, is shown in Fig. 62. Again, since we have a larger number of flows, the values in this graph are higher than in Fig. 57, but we see that with LTCP-DCR, the fraction is significantly lower than other flavors of high-speed TCP.

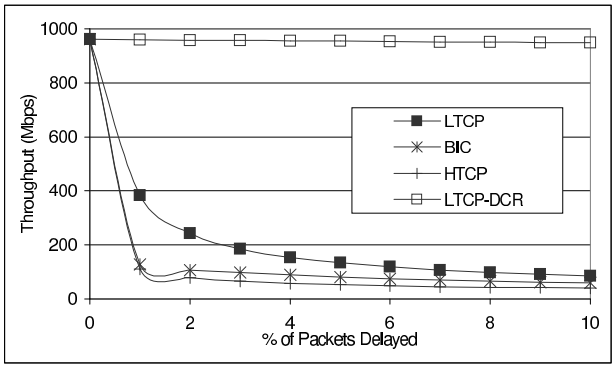


Fig. 61. Throughput in the Presence of Both Packet Reordering and Congestion in High-speed Networks

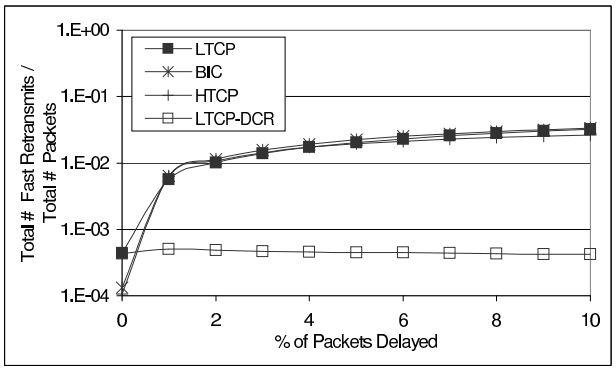


Fig. 62. Fraction of Packets That Trigger the Fast Retransmit/Recovery Algorithms

B. LTCP-PERT: Dealing with the Impact of High-speed Protocols on Buffer Size and Bottleneck Link Drop-rates

LTCP and the other proposed high-speed modifications to TCP [55]-[60] improve the capability of a single application to fully utilize the higher BDP networks. In this section, we investigate the effect of these high-speed protocols on the bottleneck link buffer usage and the packet loss rates. Our study shows the need for further refining these protocols. In general, these high-speed protocols make the probing for available bandwidth more aggressive than TCP's one-per-RTT rule in the absence of packet losses and decrease window reduction factor from TCP's cut-window-by-half rule. While this aggressiveness is useful in improving the performance in high-speed networks, if unchecked, it can result in the bottleneck buffers filling up quickly and resulting in the loss of several packets. In this section, we demonstrate that when PERT is used in conjunction with LTCP, most of the benefits of LTCP can be retained, while maintaining average link queue size low and almost eliminating packet drops due to the overflow of bottleneck link buffers.

1. Effect of High-speed Protocols on Bottleneck Buffer Usage

First, we illustrate the effect of the probing mechanisms used by the three schemes - LTCP, BIC and HTCP respectively - on the bottleneck link buffer usage through ns-2 simulations. All the three schemes are based on the loss-based TCP variants (e.g., TCP-Newreno or TCP-SACK). While the standard TCP flows increase the congestion window at the rate of one per RTT, these schemes increase the window more aggressively. Upon a packet loss, standard TCP decreases the cwnd by half, whereas these schemes propose to make the decrease less drastic. This could result in filling up the bottleneck link faster than standard TCP.

In addition, due to the nature of probing used, both LTCP and HTCP are most aggressive just before a packet loss. As a result, each packet loss event could result in the loss of several packets. BIC uses binary probing and hence its aggressiveness may be reduced while it nears the target value. However, once this value is reached, it starts to increase the window using modified slowstart. Hence, even BIC may increase its cwnd by several packets in the RTT prior to a loss, and hence may lose several packets in each loss event.

We illustrate the effects of high-speed protocols on bottleneck link buffer usage through simulations on the ns2 simulator. A simple dumb-bell topology as shown in Fig. 63 was used. The n sources (S_1 to S_n) are connected to the n receivers (R_1 to R_n) via the two routers $Router_1$ and and $Router_2$. The links connecting the sources to the router $Router_1$ and the receivers to the router $Router_2$ all have a capacity of 2.4Gbps and a delay of 10ms. The bottleneck link between the two routers has a capacity of 1Gbps and a delay of 40ms. Thus, the end-to-end RTT is 120ms. Conventionally, the bottleneck buffers are set to the delay-bandwidth product of the link. [2, 95, 96] have proposed using smaller size buffers at the routers. So, we show results with buffers equal to the delay bandwidth product (15000 packets) as well as 1/3 the delay bandwidth product (5000 packets). For simulations with more than one flow, in order to avoid phase effects and synchronization of flows, the *overhead* parameter was set to 8μs. This setting allows the source to add a random delay in the range of (0,8)μs to each outgoing packet, thus creating some randomness.

Initially, we consider a single source sending FTP traffic using LTCP, BIC or HTCP as the transport protocol to a single receiver. Since there is no cross-traffic in either direction, all packet losses are self induced due to the aggressive probing mechanism. Fig. 64 shows the instantaneous fluctuations of the queue length when the buffer size is 1/3 BDP when LTCP, BIC and HTCP are used, respectively. The

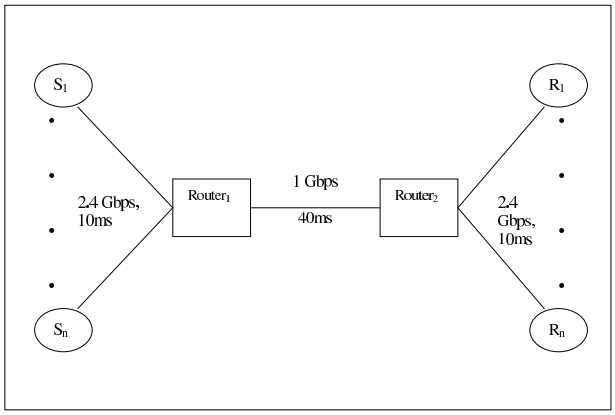


Fig. 63. Simulation Topology

buffer occupancy displays similar fluctuations when the buffer size is 1 BDP. Each oscillation resulting in the queue being full causes a 'congestion loss event' responsible for the dropping of several packets. Table XV shows a summary of the number of loss events and the corresponding packet loss rates for different protocols at different buffersizes. The measurements were recorded over 1000 seconds between 200 and 1200 seconds of each simulation. The packet loss rate for a TCP flow under similar circumstances is included for comparison.

As seen from the table, the packet loss rate in case of the high-speed protocols is significantly higher (over 3 orders of magnitude) compared to TCP. While it may be argued that this difference is due to the inability of TCP to utilize the link efficiently, we show in later sections, that using PERT with LTCP, loss rates close to 0 can be achieved, without sacrificing the other characteristics of the high-speed protocols.

Designing an aggressive probing mechanism for TCP is a challenging task. The new probing mechanism should not only ensure that the efficiency of link utilization is improved, it should also ensure that competing flows converge to fairness. Additionally, it was shown in [59] that unless carefully designed, the aggressive probing

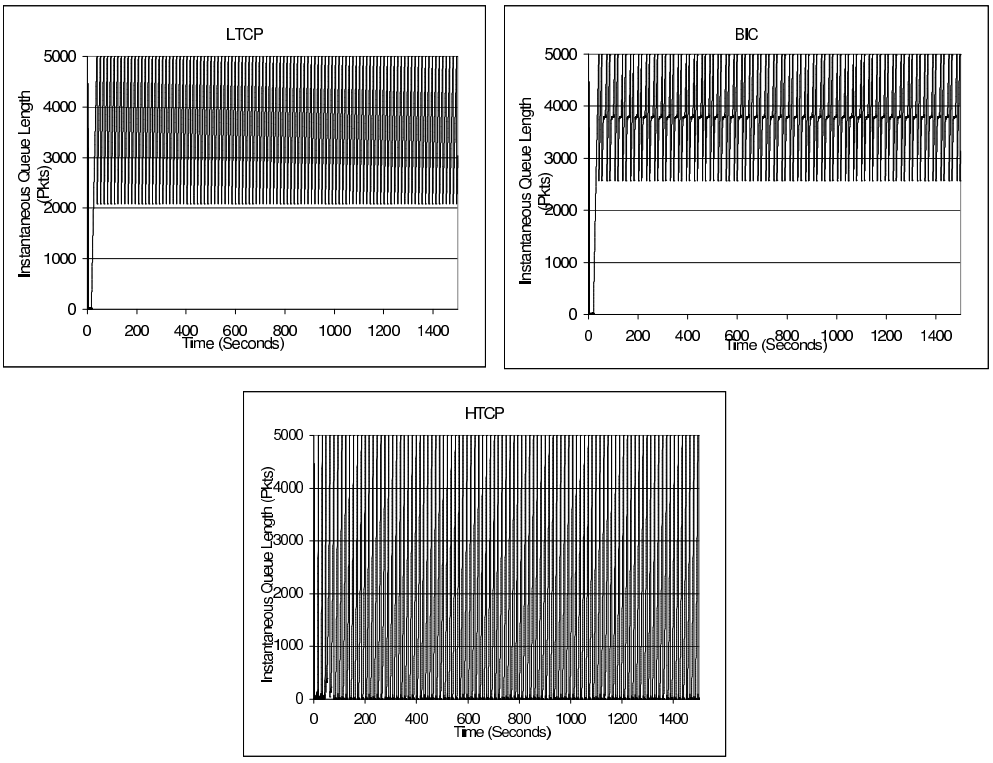


Fig. 64. Instantaneous Queue Length with Buffer Size = 5000 packets

Table XV. Summary of Congestion Loss Events and Packet Loss Rates at Different Buffer Sizes

Buffer Size = 1BDP (15000 Packets)

Protocol	# Congestion Events	Packet Loss Rate	Link Utilization (%)
TCP	-	3.08E-09	96.15
LTCP	35	1.11E-05	96.15
BIC	30	4.99E-06	96.15
HTCP	44	8.89E-05	96.15

Buffer Size = 1/3 BDP (5000 Packets)

Protocol	# Congestion Events	Packet Loss Rate	Link Utilization (%)
TCP	-	7.07E-09	89.15
LTCP	71	1.95E-05	96.15
BIC	63	1.06E-05	96.15
HTCP	66	6.34E-05	95.20

could make the RTT unfairness significantly worse than that of TCP. In LTCP, BIC [59] and HTCP[60] the authors have taken into consideration these conflicting needs and designed protocols that improve link utilization, while maintaining reasonable properties of convergence to fairness and RTT-unfairness. However, as shown by the results above, these schemes can cause significant amount of undesirable oscillations and self-induced packet losses at the bottleneck link buffers. When delayed congestion response is added to LTCP, it can further increase the loss rates by a small amount as shown in Fig. 60. In the next section, we show how PERT can be used in conjunction with LTCP for reducing the loss rates, while maintaining high link utilization and fairness among competing flows, even on high BDP links with high levels of packet reordering.

2. Offsetting the Effect of High-speed Protocols on Bottleneck Buffer Usage

In Chapter IV, we discussed how enhancing TCP with congestion prediction measure and a probabilistic response (PERT) can be used to reduce the bottleneck link drop-rates. In this section, we add PERT to the high-speed protocol LTCP to obtain a new flavor called LTCP-PERT. We show through simulations below, that LTCP-PERT can offer significant reduction in average bottleneck link queue lengths and bottleneck link drop-rates, while maintaining high link utilization in high-speed networks. For the LTCP component, we use the variable β design discussed in Section III.B. For the PERT component, the proactive window reduction is set to the same value as β used by LTCP.

The topology consists of a bottleneck link with capacity 1Gbps and delay 28ms. The number of flows sharing the link is varied from 2 to 1000 to generate different levels of congestion. The source and destination nodes are connected to the routers with links of capacity 1Gbps and delay 1ms. The bottleneck link buffersize is set to

the delay bandwidth product. Fig. 65 shows the results.

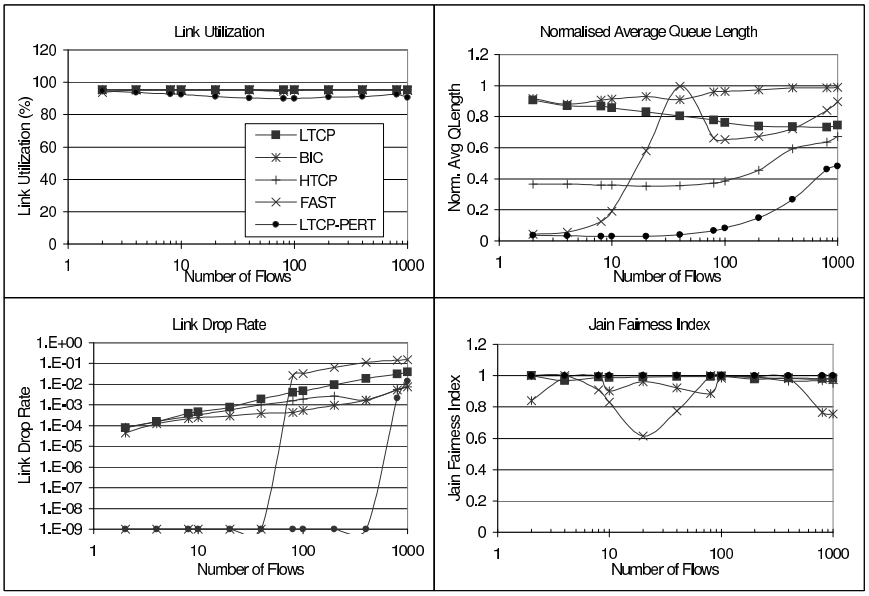


Fig. 65. Using PERT with LTCP to Reduce the Impact on Bottleneck Link Buffer Usage and Packet Drops

The figure shows four different graphs. The first graph shows the link utilization of LTCP-PERT in comparison to the other high-speed protocols. Note that, due to the use of proactive congestion avoidance, the link utilization of LTCP-PERT is slightly lower than the other protocols. However, this reduction in link utilization is very small and the trade off offers significant benefits in terms of bottleneck link usage and bottleneck link drop-rate as shown in the next two graphs. As seen from the figure, the average queue length with LTCP-PERT is significantly lower than the other protocols. While HTCP maintains lower average queue length due to variable window decrease factor, the bottleneck link drop-rates remain high and in the same ballpark as the other high-speed protocols. FAST being the only other delay-based protocol, maintains low average queue length and zero packet losses at lower number of flows.

However as the number of flows increases, since each of these flows tries to maintain a fixed number of packets in the bottleneck link queue, the average bottleneck link queue length as well as the average packet loss rate increases dramatically. In case of LTCP-PERT, the average bottleneck link queue length is maintained low even at larger number of flows. The bottleneck link packet droprates remain zero until a large number flows as well.

The benefits of using proactive congestion avoidance should not come at the cost of fairness among competing flows. The last graph in the figure shows the Jain Fairness Index with the different protocols. LTCP-PERT shows very high Jain Fairness Index (> 0.999) even for very large number of flows. Among the high-speed protocols, BIC and FAST display degradation in Jain Fairness Index, indicating that the competing flows may not always converge to fairness.

C. TCP-LDP: Putting It All Together

Finally, we investigate the benefits of putting all the components together, in the form of TCP-LDP. In order to make all the three components, LTCP, DCR and PERT perform optimally, we fine tuned each of these components. First we list the implementation details and the changes and then we look at the simulation results.

1. Implementation Details

For LTCP, we used the LTCP flavor with variable β design discussed in Section III.B. For the DCR component, we choose the window-based delay variant, where the threshold for window response is modified from 3 dupacks (used by standard LTCP) to one congestion window dupacks. For PERT, the window reduction factor was set to the same value of β_K used by LTCP instead of the fixed value of 0.35 used

in Chapter IV. A conservative metric for triggering early response was chosen. This metric is similar to the $srtt_{0.99}$ metric used in Chapter IV, with the minor difference that, whenever the instantaneous RTT falls below $srtt_{0.99}$, $srtt_{0.99}$ is reset to the value seen by the instantaneous RTT. Thus by using a large weight for the history sample, we ensure that when the RTT increases we do not respond to sporadic increases. At the same time, when the RTT decreases, we avoid unnecessary window reductions by allowing the smoothed estimate to catch up with the decreased value quickly.

Finally, using delayed response when window size is very large, may result in very bursty behavior when the reordered packet is received at the receiver and generates a cumulative ACK. In order to avoid this bursty behavior, which can be self defeating when PERT is used, we use simple packet pacing while sending the packets such that the inter-packet interval is $RTT/cwnd$ seconds.

2. Simulation Results

In this section we show the results of simulations conducted on ns-2 simulator. Note that, LTCP-DCR and TCP-LDP both try to improve the performance of the high-speed protocol LTCP in the presence of packet reordering events. The main difference between the two is that, while LTCP-DCR is a strictly loss-based protocol, TCP-LDP enhances it with the delay-based congestion avoidance to reduce the impact on bottleneck link buffer usage and the packet losses. Thus, we show in the results, the comparison of the behavior of the high-speed protocols LTCP, BIC and HTCP to that of both LTCP-DCR and TCP-LDP. The LTCP-DCR used in these simulations uses the variable β variant of LTCP discussed in Section III.B and the DCR algorithm implements the adaptive delthresh algorithm discussed in [97]. The results show the link utilization as well as the average bottleneck link queue length and packet loss rate, and the fairness among competing flows (when simulation consists of more than

one flow). Extensive results show behavior for (a) reordering only (percentage of packets delayed varied from 1e-6 to 0.1) (b) congestion only (number of flows varied from 2 to 1000) and (c) both congestion and packet reordering (similar to reordering only, but with 50 competing flows).

a. Packet Reordering Only

We first consider networks with only packet reordering. As in the previous section, packet reordering is simulated by randomly choosing packets based on a uniform distribution and delaying them. The packet delay is chosen from a normal distribution with mean 25ms and variance 8ms. The flow RTT is set to 50ms, so that the packet reordering in most cases is less than one RTT, but there in a non-zero probability that packets may be delayed by more than one RTT. The bottleneck link capacity is 1Gbps. In this simulation, since we consider the case where there is no congestion, the topology consists of only one flow. Fig. 66 shows the results as the fraction of packets delayed is varied from 1E-6 to 1E-1 resulting in potential reordering of 0.0001% to 10% of the packets.

As seen from the graph, all the high-speed TCP variants ranging from LTCP to Scalable TCP suffer severe degradation in link utilization as the fraction of packets delayed increases. In Section II.B.1, we reviewed several measurement studies that looked at the prevalence of packet reordering in the Internet. These studies indicate that the packet reordering may vary from being negligible to more than 2%. In such cases, as seen from the above graph, none of the high-speed TCP protocols will be capable of maintaining their benefits of high link utilization.

LTCP-DCR on the other hand offers very good performance. The link utilization is consistently maintained above 93% even when 10% of the packets are delayed. TCP-LDP however, does not maintain such high link utilization. The reason for this

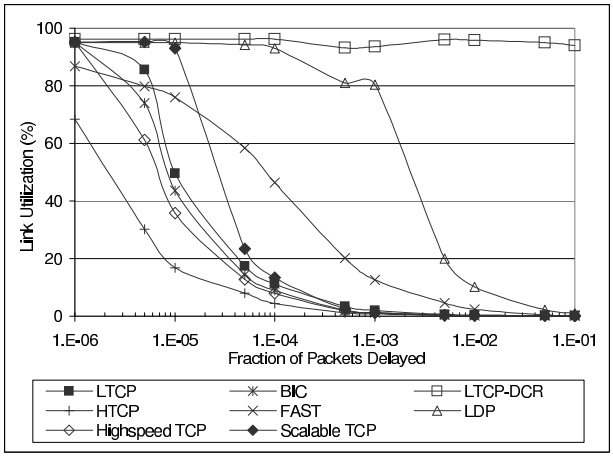


Fig. 66. Throughput in the Presence of Packet Reordering in High-speed Networks

is, when delayed congestion response is used, instead of responding with a window reduction for the receipt of three dupacks, the flow waits for the receipt of one congestion window equivalent number of dupacks. In cases of high levels of packet reordering and at large window size, this could result in a very large number of dupacks. The dupacks do not generate any valid RTT samples and hence the metric based on the smoothed RTT used by the PERT component gets stale. Proactive response based on this stale value of the smoothed RTT estimate reduces the improvement that can be obtained by using DCR.

In order to overcome this we experimented with a variant to TCP-LDP, where a proactive window reduction may not be followed by another unless at least X samples of valid RTT estimates are obtained. We used two values of X - one and 100 - and the results are presented below in Fig. 67. These protocols are notated as LDP-1Sample and LDP-100Sample respectively. As seen from the figure, waiting for at least 100 samples of valid RTT estimates before responding results in LDP behavior close to that of LTCP-DCR. However, we show in later sections where we consider

both reordering and congestion, that this may result in divergent behavior between flows. As a results, we do not recommend the use of these LDP variants.

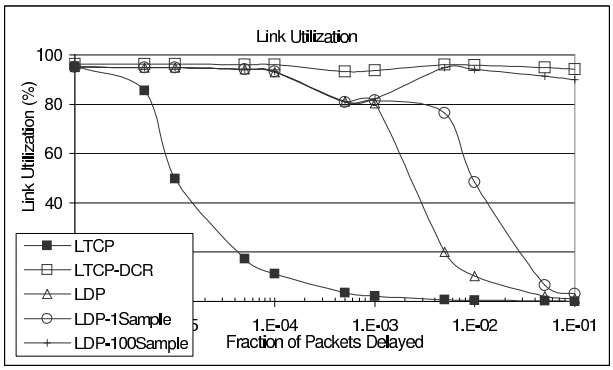


Fig. 67. Throughput of Different Variants of TCP-LDP in the Presence of Packet Reordering in High-speed Networks

b. Congestion Only

Both LTCP-DCR and TCP-LDP provide significant improvement in throughput compared to the other high-speed protocol in the presence of packet reordering. For widespread deployment, it is essential that these benefits do not come at the cost of degradation in behavior in the absence of packet reordering. In order to verify this, we conducted this experiment with only congestion in the network and no packet reordering. The bottleneck link capacity is still 1Gbps, but the number of flows sharing the link is increased from 2 to 1000 to generate different levels of congestion. Fig. 68 shows the results.

The figure shows four different graphs. In the first graph, we show the link utilization. From this graph, we see that the link utilization of LTCP-DCR remains high. However, the link utilization of TCP-LDP is slightly less than the other protocols. Next, we observe the average bottleneck link queue length. Most high-speed

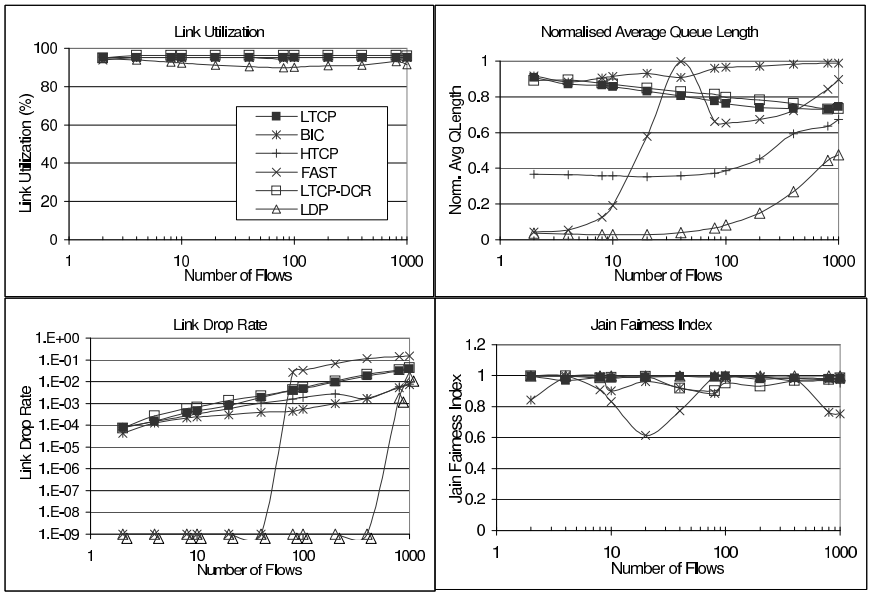


Fig. 68. Throughput in the Presence of No Packet Reordering in High-speed Networks as Congestion Is Increased

protocols, with the exception of HTCP, maintain very high average bottleneck link queue length. With LTCP-DCR, the bottleneck queue length remains similar to LTCP. With TCP-LDP, due to the use of proactive window reduction, the average bottleneck link queue usage remains very low.

Next, we examine the drop rate at the bottleneck link. The loss rate of most high-speed protocols is quite high. The loss rate with LTCP-DCR remains similar to that of LTCP, indicating that delaying congestion response by one RTT may not have any drastic impact on the bottleneck link queue usage. Among flows that use proactive response, FAST maintains zero packet losses at low number of flows. As the number of flows is increased, the packet loss rate increases drastically and is almost an order of magnitude worse than the other loss-based high-speed TCP variants. TCP-LDP on the other hand maintains zero packet losses until very large number of

flows.

Finally, we observe the fairness among the different flows sharing the bottleneck link. TCP-LDP exhibits very good fairness behavior with the Jain Fairness Index consistently remaining more than 0.999. LTCP-DCR exhibits reasonably good fairness behavior with the Jain Fairness Index remaining greater than 0.95 in most cases and the worst behavior observed still had Jain Fairness Index of 0.9.

c. Both Packet Reordering and Congestion

Next, we examine how the different protocols behave when the network consists of both packet reordering and packet losses due to congestion. This experiment is similar to the experiment with only packet reordering, except that the bottleneck link is now shared by 50 flows. Fig. 69 shows the results.

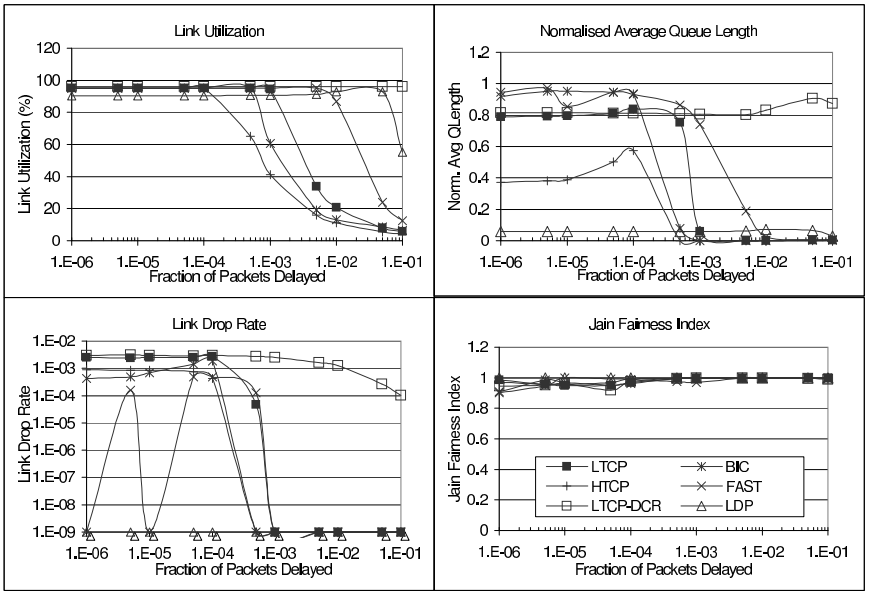


Fig. 69. Throughput in the Presence of Both Packet Reordering and Losses due to Congestion

Since 50 flows share the bottleneck link, the aggregate link utilization is improved. However, at high levels of packet reordering, the degradation, is still very drastic. In case of LTCP-DCR, the link utilization is maintained high even for high levels of packet reordering. In case of TCP-LDP, the link utilization is around 90% when packet reordering is less than 5%. Beyond that, the link utilization decreases, but the degradation is not as drastic as in the case of other high-speed flows.

As discussed earlier, only TCP-LDP can maintain low average bottleneck link queue and bottleneck link drop-rates, when the link is not under utilized. The average bottleneck link queue length and drop-rate of LTCP-DCR is similar to that of LTCP when LTCP it is capable of fully utilizing the link. The Jain Fairness Index of all the protocols remains high.

We now revisit the two new variants of TCP-LDP namely, LDP-1Sample and LDP-100Sample that we discussed during the simulations with packet reordering only. Fig. 70 shows the results with these new flavors for the above experiment.

As expected, both the variants improve the link utilization of TCP-LDP. However, the results for average bottleneck link queue length, loss rates and Jain Fairness Index are not as favorable. Further inspection of these cases revealed the following. Consider two flows, one operating at very high congestion window and the other operating at low congestion window. The flow at higher congestion window sees more packet reordering events. Subsequently, due to the receipt of many dupacks which do not provide a usable RTT estimate, the proactive response is temporarily suspended. The congestion window thus has a chance to grow to a larger value, where it has a possibility of seeing higher packet reordering events (for the same percentage of packets delayed) and so on. On the other hand, for the flow with smaller window, for the same given rate at which packets are reordered, the number of events observed by the flow is smaller, compared to the larger flow. So most of the acknowledgments received

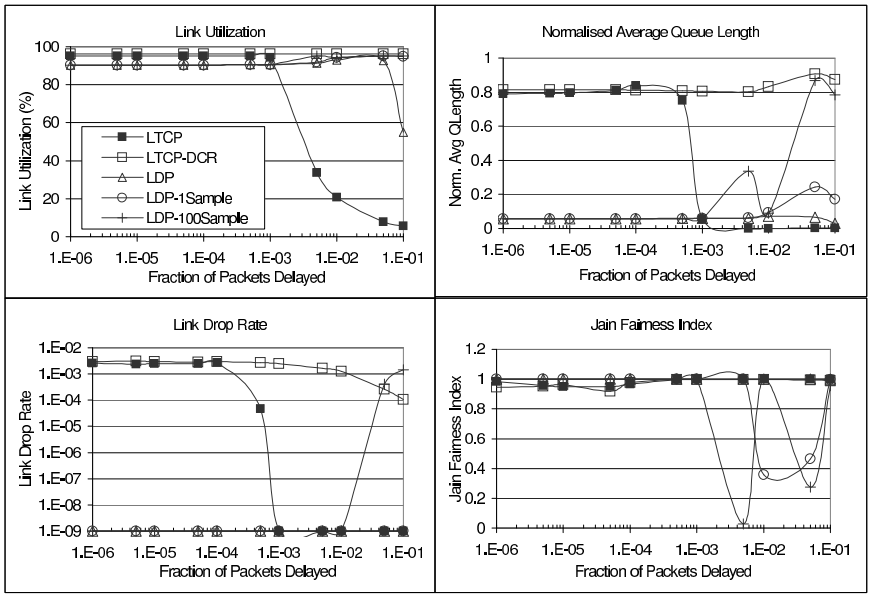


Fig. 70. Throughput in the Presence of Both Packet Reordering and Congestion for Different Variants of TCP-LDP

result in useful RTT estimates. This will cause the proactive response to remain ON, and hence window may get decreased, if the bottleneck link queue length increases. Thus the two flows that started at different window sizes continue to diverge and result in unfavorable fairness characteristics. As a result, we do not recommend the use of these TCP-LDP variants. The basic TCP-LDP variant discussed in Section 1 does not have this problem and as a result displays excellent fairness characteristics.

d. Simulation with More Complex Topologies

In this experiment we verify that the benefits of LTCP-DCR and LDP are available in complex topologies as well. The network in this simulation consists of four bottleneck links between five routers. Each router is connected to a cloud of nodes. Traffic goes from one cloud to the other in the directions as shown in Fig. 71. The traffic consists of a mix of 20 long-term flows and 100 web-sessions. The router R3 simulates a mis-

configured router and results in reordering 1% of the packets passing through it. The delay used for reordering the packets uses the same model as before.

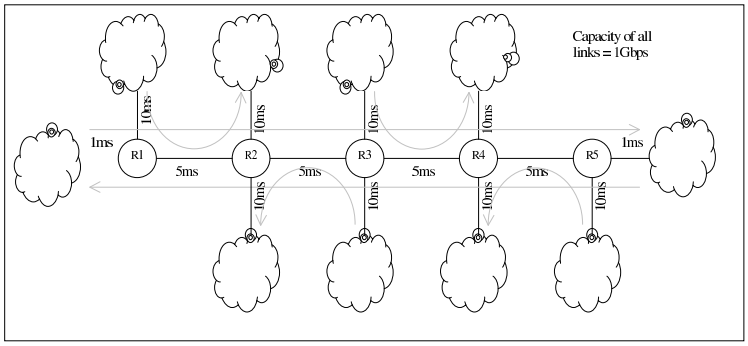


Fig. 71. Topology with Multiple Bottleneck Links, Forward as Well as Reverse Traffic, and Long-term as Well as Short Web-like Flows

Fig. 72 shows the link utilization, drop-rates on each of the bottleneck links, normalized average bottleneck link queue length and the Jain Fairness Index of the long-term flows passing through each set of routers. From the figure we see that the utilization of LTCP, BIC and HTCP see drastic degradation on the links associated with the 'mis-configured' router R3. While the utilization for FAST is not as drastically reduced, it is still quite low. Both LTCP-DCR and LDP, however manage to maintain the link utilization high.

The drop-rate on the links R1-R2 and R5-R4, which do not see degradation due to packet reordering, is similar for LTCP and LTCP-DCR. On links R3-R2 and R3-R4 that suffer underutilization, LTCP sees no packet losses. But since LTCP-DCR flows do not underutilize these links they result in packet losses due to congestion. Note that the packet reordering model is probabilistic and not all reordering events can be avoided - as a result the packet drop-rate on these links is lower than those that do not have any reordering. With FAST and LDP, the bottleneck link drop rate is 0,

since the proactive delay-based mechanism reduces the congestion window *before* a packet drop can occur.

On the routers associated with links that are not underutilized, the normalized average queue length LTCP, BIC and LTCP-DCR is very high, indicating that these queues remain close to full most of the time. In case of HTCP, the normalized queue length is slightly lower even though the drop-rates are similar to BIC. This is because, HTCP uses variable window reduction in the range $(0.5, 0.8)$, allowing the queue to flush out every time a packet drop occurs. With FAST, the average link queue length is around 50%. However, LDP maintains the average queue length less than 5%, without significantly impacting the link utilization.

Finally, we look at the Jain Fairness Index of all the the long-term flows between each sets of source/destination. From the table, we see that the Jain Fairness Index for both LTCP-DCR and LDP remains high.

D. Conclusion

In this chapter, we evaluated the different schemes proposed in this dissertation, when then are combined with each other. We first inspected the combination of LTCP and DCR, called LTCP-DCR, to showed that it offers significant performance improvements in high-speed networks in the presence of packet reordering. We then added the delay based component PERT to this to obtain TCP-LDP. While TCP-LDP can reduce the average queue length of the bottleneck link drop-rates and almost entirely eliminate the bottleneck link drops, the improvement in link utilization is not as high as LTCP-DCR. This can be attributed to the fact that the PERT component requires RTT samples at relatively high frequency whereas the DCR component results in a large number of dupacks which do not generate any RTT samples. Due to the

	LTCP	BIC	HTCP	FAST	LTCP-DCR	LDP
Link Utilization						
R1-R2	96.87	96.41	94.25	95.35	96.52	92.48
R3-R2	13.29	5.41	7.96	60.96	96.65	95.45
R3-R4	11.01	9.59	8.96	59.09	95.19	94.62
R5-R4	94.65	96.53	93.91	94.92	95.56	91.64
Link Droprate						
R1-R2	1.89E-03	7.37E-04	5.20E-04	0	2.57E-03	0
R3-R2	0	0	0	0	2.83E-04	0
R3-R4	0	0	0	0	2.51E-04	0
R5-R4	1.66E-03	7.23E-04	4.30E-04	0	2.41E-03	0
Normalised Average Queue Length						
R1-R2	0.94	0.97	0.68	0.53	0.93	0.03
R3-R2	0.00	0.00	0.00	0.01	0.89	0.04
R3-R4	0.00	0.00	0.00	0.01	0.89	0.04
R5-R4	0.94	0.97	0.71	0.52	0.93	0.03
Jain Fairness Index						
R1-R5	0.9834	0.9993	0.9966	0.9951	0.9364	0.9999
R1-R2	0.9585	0.9666	0.9518	0.6954	0.9879	1.0000
R3-R2	0.9994	0.9986	0.9986	0.9997	0.9956	0.9976
R3-R4	0.9994	0.9987	0.9994	0.9994	0.9938	0.9962
R5-R4	0.9750	0.9671	0.9320	0.7121	0.9854	1.0000
R5-R1	0.9899	0.9950	0.9985	0.9929	0.9615	1.0000

Fig. 72. Link Utilization, Link Drop-rate and Jain Fairness Index for Flows on a Multiple Bottleneck Link Router with a Mis-configured Router That Causes Packet Reordering

delay based component (PERT) used in TCP-LDP, it will not be able to compete with purely loss-based schemes and hence it may not be suitable for wide spread deployment. It is a good candidate for networks where all the traffic uses some for of proactive congestion avoidance scheme. In such an environment, TCP-LDP offers relatively high link utilization in the presence of packet reordering, while keeping the bottleneck link queues and the loss due to bottleneck link buffer overflows low.

LTCP-DCR on the other hand is a purely loss-based scheme. We have conducted extensive simulations with it and noted that it increases the link utilization significantly in the presence of packet reordering, while having minimal impact on fairness characteristics in the presence of congestion. As a result, we believe that it may be safe for widespread deployment in high-speed networks.

REFERENCES

[1] M. Duke, R. Braden, W. Eddy, and E. Blanton, "A roadmap for TCP specification documents," Internet Draft, Internet Engineering Task Force, February 2006, Status as of June 2006 : Approved by IESG, awaiting processing and publishing. Available: http://www.ietf.org/internet-drafts/draft-ietf-tcpm-tcp-roadmap-06.txt; Accessed: June 05, 2006.

[2] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proc. ACM SIGCOMM*, August/September 2004, pp. 281–292.

[3] V. Paxson, "End-to-end Internet packet dynamics," *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 277–292, June 1999.

[4] L. Gharai, C. Perkins, and T. Lehman, "Packet reordering, high speed networks and transport protocol performance," in *Proc. ICCCN*, October 2004, pp. 73–78.

[5] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A reordering-robust TCP with DSACK," in *Proc. IEEE ICNP*, November 2003, pp. 95–106.

[6] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, December 1997.

[7] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," *ACM Computer Communication Review*, vol. 32, no. 1, pp. 20–30, January 2002.

[8] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. ICDCS*, May 1995, pp. 126–143.

[9] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bhargavan, "WTCP: A reliable transport protocol for wireless wide-area networks," in *Proc. ACM MOBICOM*, August 1999, pp. 231–241.

[10] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's loss recovery using limited transmit," RFC 3042, Internet Engineering Task Force, January 2001, Available: http://www.ietf.org/rfc/rfc3042.txt; Accessed: June 05, 2006.

[11] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," RFC 2581, Internet Engineering Task Force, April 1999, Available: http://www.ietf.org/rfc/rfc2581.txt; Accessed: June 05, 2006.

[12] R. Han and D. G. Messerschmitt, "A progressively reliable transport protocol for interactive wireless multimedia," *ACM/Springer-Verlag Multimedia Systems Journal*, vol. 7, no. 2, pp. 141–156, March 1999.

[13] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM*, September 1998, pp. 303–314.

[14] D. Bansal and H. Balakrishnan, "Binomial congestion control algorithms," in *Proc. IEEE INFOCOM*, April 2001, pp. 631–640.

[15] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM*, August 2000, pp. 43–56.

[16] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker, "Dynamic behavior of slowly responsive congestion control algorithms," in *Proc. ACM SIGCOMM*, August 2001, pp. 263–274.

[17] D. Loguinov and H. Radha, "End-to-end Internet video traffic dynamics: Statistical study and analysis," in *Proc. IEEE INFOCOM*, June 2002, pp. 723–732.

[18] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Measurement and classification of out-of-sequence packets in a tier-1 IP backbone," in *Proc.*

IEEE INFOCOM, March 2003, pp. 1199–1209.

[19] Y. Wang, G. Lu, and X. Li, "A study of Internet packet reordering," in *Proc. ICOIN*, February 2004, pp. 350–359.

[20] J. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pathological network behavior," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 789–798, December 1999.

[21] X. Zhou and P. V. Mieghem, "Reordering of IP packets in Internet," in *Proc. PAM*, April 2004, pp. 237–246.

[22] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An extension to the selective acknowledgement (SACK) option for TCP," RFC 2883, Internet Engineering Task Force, July 2000, Available: http://www.ietf.org/rfc/rfc2883.txt; Accessed: June 05, 2006.

[23] R. Ludwig and M. Meyer, "The Eifel detection algorithm for TCP," RFC 3522, Internet Engineering Task Force, April 2003, Available: http://www.ietf.org/rfc/rfc3522.txt; Accessed: June 05, 2006.

[24] *ns-2 Network Simulator*, Available: http://www.isi.edu/nsnam/; Accessed: June 05, 2006.

[25] R. Yavatkar and N. Bhagawat, "Improving end-to-end performance of TCP over mobile internetworks," in *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, December 1994, pp. 146–152.

[26] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM Computer Communications Review*, vol. 27, no. 5, pp. 19–43, October 1997.

[27] K.-Y. Wang and S. K. Tripathi, "Mobile-end transport protocol: An alternative to TCP/IP over wireless links," in *Proc. IEEE INFOCOM*, March/April 1998,

pp. 1046–1053.

[28] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, "Improving TCP/IP performance over wireless networks," in *Proc. ACM MOBICOM*, November 1995, pp. 2–11.

[29] H. M. Chaskar, T. V. Lakshman, and U. Madhow, "TCP over wireless with link level error control: Analysis and design methodology," *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 605–615, October 1999.

[30] H. Balakrishnan and R. H. Katz, "Explicit loss notification and wireless web performance," Presented at IEEE Globecom Internet Mini-Conference, Sydney, Australia, November 1998.

[31] K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC 2481, Internet Engineering Task Force, January 1999, Available: http://www.ietf.org/rfc/rfc2481.txt; Accessed: June 05, 2006.

[32] R. Krishnan, M. Allman, C. Partridge, and J. P. G. Sterbenz, "Explicit transport error notification for error-prone wireless and satellite networks," Tech. Rep. 8333, BBN Technologies, February 2002, Available: http://www.ir.bbn.com/projects/pace/eten/; Accessed: June 05, 2006.

[33] N. H. Vaidya, M. Mehta, C. Perkins, and G. Montenegro, "Delayed duplicate acknowledgement: a TCP-unaware approach to improve performance of TCP over wireless," *Journal of Wireless Communications and Mobile Computing*, vol. 2, no. 1, pp. 59–70, February 2002.

[34] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," RFC 2018, Internet Engineering Task Force, October 1996, Available: http://www.ietf.org/rfc/rfc2018.txt; Accessed: June 05, 2006.

[35] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. ACM MOBICOM*, July 2001, pp. 287–297.

[36] D. Eckhardt and P. Steenkiste, "Improving wireless lan performance via adaptive local error control," in *Proc. IEEE ICNP*, October 1998, pp. 327–338.

[37] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over satellite channels using standard mechanisms," RFC 2488, Internet Engineering Task Force, January 1999, Available: http://www.ietf.org/rfc/rfc2488.txt; Accessed: June 05, 2006.

[38] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance enhancing proxies intended to mitigate link-related degradations," RFC 3135, Internet Engineering Task Force, June 2001, Available: http://www.ietf.org/rfc/rfc3135.txt; Accessed: June 05, 2006.

[39] S. Floyd, "Congestion control principles," RFC 2914, Internet Engineering Task Force, September 2000, Available: http://www.ietf.org/rfc/rfc2914.txt; Accessed: June 05, 2006.

[40] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *Proc. ACM SIGCOMM*, August 1996, pp. 117–130.

[41] L. Vicisano, L. Rizzo, and J. Crowcroft, "TCP-like congestion control for layered multicast data transfer," in *Proc. IEEE INFOCOM*, March/April 1998, pp. 996–1003.

[42] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP buffer tuning," in *Proc. ACM SIGCOMM*, August 1998, pp. 315–323.

[43] E. Weigle and W. Feng, "Dynamic right-sizing: a simulation study," in *Proc. IEEE ICCCN*, October 2001, pp. 152–158.

[44] B. L. Tierney, D. Gunter, J. Lee, M. Stoufer, and J. B. Evans, "Enabling network-aware applications," in *Proc. IEEE HPDC*, August 2001, pp. 281–302.

[45] T. Dunigan, M. Mathis, and B. Tierney, "A TCP tuning daemon," in *Proc. ACM/IEEE Conference on Supercomputing*, November 2002, pp. 1–16.

[46] E. Weigle and W. Feng, "A comparison of TCP automatic tuning techniques for distributed computing," in *Proc. IEEE HPDC*, July 2002, pp. 265–275.

[47] S. Ostermann, M. Allman, and H. Kruse, "An application-level solution to TCP's satellite inefficiencies," Presented at Workshop on Satellite-based Information Services (WOSBIS), Rye, NY, November 1996.

[48] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke, "Applied techniques for high bandwidth data transfers across wide area networks," Presented at International Conference on Computing in High Energy and Nuclear Physics, Beijing, China, September 2001.

[49] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC storage resource broker," in *Proc. CASCON'98 Conference*, December 1998, p. 5.

[50] R. Long, L. E. Berman, L. Neve, G. Roy, and G. R. Thoma, "An application-level technique for faster transmission of large images on the Internet," in *Proc. SPIE: Multimedia Computing and Networking*, February 1995, pp. 116–129.

[51] H. Sivakumar, S. Bailey, and R. Grossman, "PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks," in *Proc. SuperComputing*, November 2000, p. 37.

[52] J. Crowcroft and P. Oechslin, "Differentiated end-to-end Internet services using a weighted proportional fair sharing TCP," *ACM Computer Communications Review*, vol. 28, no. 3, pp. 53–67, July 1998.

[53] T. Hacker, B. Noble, and B. Athey, "Improving throughput and maintaining fairness using parallel TCP," in *Proc. IEEE INFOCOM*, March 2004, pp. 2480–2489.

[54] H. Hsieh and R. Sivakumar, "pTCP: An end-to-end transport layer protocol for striped connections," in *Proc. IEEE ICNP*, November 2002, pp. 24–33.

[55] S. Floyd, "Highspeed TCP for large congestion windows," RFC 3649, Internet Engineering Task Force, December 2003, Available: http://www.ietf.org/rfc/rfc3649.txt; Accessed: June 05, 2006.

[56] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *ACM Computer Communications Review*, vol. 33, no. 2, pp. 83–91, April 2003.

[57] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in *Proc. IEEE INFOCOM*, March 2004, pp. 2490–2501.

[58] L. Brakmo, S. O'Malley, and L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," in *Proc. ACM SIGCOMM*, August 1994, pp. 24–35.

[59] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks," in *Proc. IEEE INFOCOM*, March 2004, pp. 2514–2524.

[60] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proc. PFLDNet'04*, February 2004, p. 5.

[61] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. ACM SIGCOMM*, August 2002, pp. 89–102.

[62] M. R. Meiss, "Tsunami: A high-speed rate-controlled protocol for file transfer," Tech. Rep., Indiana University, Available: http://steinbeck.ucs.indiana.edu/~mmeiss/papers/tsunami.pdf; Accessed: June 05, 2006.

[63] E. He, J. Leigh, O. Yu, and T. A. DeFanti, "Reliable Blast UDP : Predictable high performance bulk data transfer," in *Proc. IEEE Cluster Computing*, September 2002, pp. 317–324.

[64] R. Grossman, M. Mazzucco, H. Sivakumar, Y. Pan, and Q. Zhang, "Simple available bandwidth utilization library for high-speed wide area networks," *Journal of Supercomputing*, vol. 34, no. 3, pp. 231–242, December 2005.

[65] R. X. Wu and A. A. Chien, "GTP: Group transport protocol for lambda-grids," in *Proc. IEEE/ACM International Symposium on Cluster Computing and the Grid*, April 2004, pp. 228–238.

[66] S. Floyd, "Limited slow-start for TCP with large congestion windows," RFC 3742, Internet Engineering Task Force, March 2004, Available: http://www.ietf.org/rfc/rfc3742.txt; Accessed: June 05, 2006.

[67] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFC 1323, Internet Engineering Task Force, May 1992, Available: http://www.ietf.org/rfc/rfc1323.txt; Accessed: June 05, 2006.

[68] S. J. Golestani and K. K. Sabnani, "Fundamental observations on multicast congestion control in the Internet," in *Proc. IEEE INFOCOM*, March 1999, pp. 982–989.

[69] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," in *Proc. Computer Networks and ISDN Systems*, June 1989, pp. 1–14.

[70] P. Sarolahti and A. Kuznetsov, "Congestion control in linux TCP," in *Proc. USENIX*, June 2002, pp. 49–62.

[71] *Iperf Version 1.7.0*, Available: http://dast.nlanr.net/Projects/Iperf/; Accessed: June 05, 2006.

[72] *PingER History Tables at SLAC*, Available: http://www-iepm.slac.stanford.edu/cgi-wrap/table.pl?from=Country&to=Cou%ntry&file=average_rtt&date=2006-06; Accessed: June 05, 2006.

[73] S. Floyd and V. Jacobson, "Random early detection gateways for congestion control," *IEEE/ACM Transactions on Networking*, vol. 1, no.4, pp. 397–412, August 1993.

[74] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin, "REM: Active queue management," *IEEE Network*, vol. 15m no. 3, pp. 48–53, May/June 2001.

[75] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *Proc. IEEE INFOCOM*, April 2001, pp. 1726–1734.

[76] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual queue algorithm for active queue management," in *Proc. ACM SIGCOMM*, August 2001, pp. 123–134.

[77] W. Feng, D. Kandlur, D. Saha, and K. Shin, "The BLUE queue management algorithms," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 513–528, August 2002.

[78] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the internet," in *Proc. IWQoS*, June 2005, pp. 271–285.

[79] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One more bit is enough," in *Proc. ACM SIGCOMM*, August 2005, pp. 37–48.

[80] Y. Zhang, D. Leonard, and D. Loguinov, "JetMax: Scalable max-min congestion control for high-speed heterogeneous networks," in *To appear in Proc. IEEE INFOCOM*, April 2006.

[81] R. Jain, "A delay based approach for congestion avoidance in interconnected heterogeneous computer networks," *ACM Computer Communication Review*, vol. 19, no. 5, pp. 56–71, October 1989.

[82] Z. Wang and J. Crowcroft, "A new congestion control scheme: Slow start and search (Tri-S)," *ACM Computer Communication Review*, vol. 21, no. 1, pp. 32–43, January 1991.

[83] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in 4.3–Tahoe BSD TCP congestion control," *ACM Computer Communication Review*, vol. 22, no. 2, pp. 9–16, April 1992.

[84] J. Martin, A. Nilsson, and I. Rhee, "Delay-based congestion avoidance for TCP," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 356–369, June 2003.

[85] R. S. Prasad, M. Jain, and C. Dovrolis, "On the effectiveness of delay-based congestion avoidance," in *Proc. PFLDNet 2004*, February 2004, p. 3.

[86] S. Rewaskar, J. Kaur, and D. Smith, "Why don't delay-based congestion estimators work in the real-world?," Tech. Rep. TR06-001, Department of Computer Science, UNC Chapel Hill, July 2005.

[87] A. A. Awadallah and C. Rai, "TCP-BFA: Buffer fill avoidance," in *Proc. IFIP High Performance Networking Conference*, September 1998, pp. 575–594.

[88] M. C. Weigle, K. Jeffay, and F. Donelson Smith, "Delay-based early congestion detection and adaptation in TCP: impact on web performance," *Computer Communications*, vol. 28, no. 8, pp. 837–850, May 2005.

[89] S. Biaz and N. Vaidya, "Is the round-trip time correlated with the number of packets in flight?," in *Proc. IMC*, October 2003, pp. 273–278.

[90] M. Roughan, "Fundamental bounds on the accuracy of network performance measurements," in *Proc. ACM SIGMETRICS*, June 2005, pp. 253–264.

[91] W. Feng, D. Kandlur, D. Saha, and K. Shin, "A self-configuring RED gateway," in *Proc. IEEE INFOCOM*, March 1999, pp. 1320–1328.

[92] S. Floyd and V. Paxson, "Difficulties in simulating the Internet," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 392–403, August 2001.

[93] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *Proc. ACM SIGCOMM*, September 1999, pp. 301–313.

[94] A. Kuzmanovic and E. W. Knightly, "TCP-LP: A distributed algorithm for low priority data transfer," in *Proc. IEEE INFOCOM*, April 2003, pp. 1691–1701.

[95] D. Barman, G. Smaragdakis, and I. Matta, "The effect of router buffer size on highspeed TCP performance," in *Proc. IEEE Globecom*, November 2004, pp. 1617–1621.

[96] S. Gorinsky, A. Kantawala, and J. Turner, "Link buffer sizing: A new look at the old problem," in *Proc. ISCC 2005*, June 2005, pp. 507–514.

[97] S. Bhandarkar, A. L. N. Reddy, M. Allman, and E. Blanton, "Improving the robustness of TCP to non-congestion events," Internet Draft, Internet Engineering Task Force, January 2006, Status as of June 2006 : Approved by IESG, awaiting processing and publishing. Available: http://www.ietf.org/internet-drafts/draft-ietf-tcpm-tcp-dcr-07.txt; Accessed: June 05, 2006.

VITA

Sumitha Bhandarkar received her B.E. degree in electronics and communications engineering from Sri Jayachamarajendra College of Engineering, Mysore University, India in 1997 and her M.S. and Ph.D. in computer engineering from the Texas A&M University, College Station, TX, in 2001 and 2006 respectively. Her research interests are in the areas of high-performance networking, Internet congestion control and wireless networks. She has been a recipient of the National Merit Scholarship from the Govt. of India during 1991-1997 and the Schlumberger "Faculty for the Future" fellowship during 2004-2005. She has worked as a senior software engineer at Tata Unisys Ltd. (India) from 1997-1999 and as a summer/fall Intern at Kiyon Inc. in 2004. She is a student member of the IEEE and the ACM computer society and participates in the standardization work at IETF as part of the TCPM working group. She can be contacted at the following address : Department of Electrical and Computer Engineering, Texas A&M University, 214 Zachry Engineering Center, College Station, TX 77843-3128.

The typist for this thesis was Sumitha Bhandarkar.