ABSTRACT

Ranjith S. Jayaram. Performance Evaluation of TEAR, a TCP-friendly Flow Control Protocol, Over the Internet and Wireless Networks. (Under the direction of Professor Injong Rhee.)

TCP Emulation at Receivers (TEAR) is a TCP-friendly protocol that has been proposed for real-time multimedia flow control. Most best-effort traffic on the Internet is well-served by TCP, the dominant transport protocol. However, many applications with real-time constraints, such as multimedia streaming find TCP's response to congestion quite severe and too drastic to deliver acceptable end-user quality. TEAR was designed in order to provide smoothly varying rate changes for such applications while being friendly to competing TCP flows. In this thesis, weevaluate and verify TEAR's performance over the Internet. We verify TEAR's fairness to TCP, the smoothness of its rate fluctuations and its stability in the presence of network perturbations. We then adapt TEAR to run over wireless networks and consider using round-trip delay instead of packet loss as a congestion indication for wireless networks. We present the results of our experiments with TEAR over commercially deployed wireless networks in South Korea. We recount our experiences with the loss-delay characteristics of these networks. We analyze how TEAR competes with TCP, which is known to suffer from severe degradations in environments where the underlying network is unreliable. We then study TEAR's rate variations and the increased longer-term predictability it provides over wireless networks. Finally, we compare the performance of a reliable protocol we built over TEAR with TCP.

Performance Evaluation of TEAR, a TCP-friendly Flow Control Protocol, Over the Internet and Wireless Networks

by

Ranjith S. Jayaram

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, NC

July 2001

APPROVED BY:

George N. Rouskas

Wenke Lee

Injong Rhee, Chair of Advisory Committee

To

Amma, Appa, and Anna.

BIOGRAPHY

Ranjith S. Jayaram was born in Alapuzha, India in 1976. He received his B.Tech degree in Electronics and Communication Engineering from the University of Calicut in 1997. He joined the graduate program in the Computer Science department at North Carolina State University in 1999 and is currently working towards the completion of a M.S. in Computer Science.

ACKNOWLEDGEMENTS

I would like to thank Dr. Injong Rhee for supporting me in the role of an advisor during the course of my thesis research and for his invaluable encouragement during my stay at NCSU. Working with him has been a highly rewarding experience.

I'm indebted to Kyungtae Woo and Woo-il Kwon for their help in conducting the experiments over wireless networks.

No amount of thanks would be too much for my roommates Ajith, Arvind and Parag for their invaluable support and incredible tolerance when I continuously missed my cooking turns while writing this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Transmission Control Protocol (TCP) [19][1][3] is the dominant protocol of the Internet. It is well known that the stability of the Internet depends on the end-to-end congestion control mechanisms of TCP. The Additive Increase Multiplicative Decrease (AIMD) algorithm used by TCP has been one of the key contributors to the success, widespread deployment and viability of the Internet.

It has been proved that all classes of AIMD congestion control algorithms are stable [10]. For TCP, the sending rate is controlled by a congestion window which is halved for every window of data containing a packet drop, and increased by roughly one packet per window of data otherwise. The decrease term of 1/2 was selected because the performance penalty for this term is smaller compared to other terms, for instance 7/8 used in DECBit, which is also based on AIMD. The multiplicative factor of 2 results in frequent rate fluctuations and bursty transmission for TCP. While this is acceptable for applications such as bulk data transfer, applications which have real-time constraints such as streaming multimedia find this inadmissible. In particular, TCP's halving of its sending rate to single packet losses is quite severe on these applications and degrades their performance below acceptable user levels. Such applications require a more smoothly-changing sending rate than that of TCP. Many of the currently available multimedia applications solve this problem simply by sending at a constant rate. These greedy applications might be able to deliver acceptable quality for their users at times, but they pose a grave danger to the stability of the Internet and are primary causes of concern for a congestion collapse [6]. Applications that are sensitive to drastic rate fluctuations must instead use congestion control schemes which address the need of providing smoothly varying rate changes and at

the same time compete fairly with TCP to ensure network stability.

TCP Emulation at Receivers (TEAR) is a congestion control protocol that was proposed in [24]. TEAR estimates TCP friendly rates by observing loss events and emulating TCP's flow control behavior at the receivers. The protocol shifts most congestion control mechanisms to the receivers and the rates estimated by the receivers can be used to control the rates of non-TCP flows. TEAR is suitable for streaming multimedia applications because its rate variations are smooth and due to the TCP emulation, it is also TCP friendly. It provides a viable mechanism to provide relatively smooth congestion control for real-time traffic. The cost of this is a more moderate response to transient changes in congestion.

Unlike TCP, receivers are endowed with intelligence in the TEAR architecture. The receiver does not blindly send back acknowledgments and let the sender take congestion control decisions. Instead, it tries to make sense out of the congestion indications that it observes - packet losses or delays. These congestion signals are used by the receiver to maintain a congestion window, emulate the TCP sender's flow control functions and determine its own appropriate receiving rate. TEAR uses the concept of 'rounds' and 'epochs' to smoothen out extreme variations and thus avoid undesirable rate fluctuations. This smoothening means that TEAR does not react to individual packet losses, i.e., there is no drastic rate adaptation over fine timescales. The design also ensures that TEAR responds effectively to persistent congestion, i.e., network congestion over longer time periods.

One of the goals of this thesis is to evaluate and verify the performance of TEAR over the real Internet. The objectives include investigating whether TCP Emulation indeed does result in TCP friendliness, how TEAR flows compete with TCP, how TEAR flows compete with each other and finding out how effective TEAR is in estimating the available bandwidth and converging to the fair-share bandwidth in steady state conditions. We find that TEAR is friendly to TCP and offers superior rate smoothness while achieving comparable throughput. TEAR flows compete fairly with each other and with competing TCP flows. The protocol is also shown to be stable, in the sense that in the presence of network transients, eventually TEAR flows reach their fair share of the bandwidth when the network reaches steady state.

A second goal of the thesis is to study the performance of TEAR on wireless networks and compare its performance with TCP. Wireless networks have characteristics that are distinct from the wired Internet. On the Internet, transport protocols can safely

assume a reliable underlying network in which packet losses are due to congestion. However, in a wireless network, packet losses tend to occur more often due to wireless link unreliability than due to congestion. When TCP is used over wireless links, each packet loss is treated as an indication of incipient network congestion and the congestion control mechanisms of TCP are invoked. This severely degrades TCP throughput on wireless networks. We investigate how TEAR can use the congestion signals the receiver observes to improve over TCP's performance. In particular we make a modification to TEAR to use round-trip delays as an indication of congestion instead of packets losses and study the effect of this on throughput and rate smoothness.

There is an interesting contrast between the use of TEAR over the Internet and the wireless environments. On the Internet, for flows that require reliable data transmission, TCP is still the best protocol to use. It is applications that need to maintain a smoothly varying sending rate and networks that need to be guarded against unresponsive, bandwidth hungry applications that benefit from the use of TCP friendly unreliable congestion control mechanisms like TEAR. On the wired Internet, the emphasis is thus on slowly changing rates and TCP friendliness.

On the other hand, in wireless networks, TCP is often not the best protocol to use even for reliable data transmission. In many commercially deployed wireless networks, because some amount of per-channel provisioning is present, TCP friendliness becomes less of a consideration compared to effective throughput. Also, since TCP underperforms anyway over wireless, attaching more value of TCP friendliness in such environments is not fully justified. We investigate the flexibility offered by TEAR to adapt to these differing requirements of providing smoothly changing TCP friendly rates on the Internet and delivering higher throughput on wireless networks while not starving TCP flows.

We find that TEAR achieves better throughput than TCP over wireless networks and remains friendly to TCP, in the sense that it does not drive TCP to starvation. The use of round-trip times instead of packet losses for congestion indication results in more predictable variations in TEAR's sending rate and also in fewer rate fluctuations. This longer-term predictability can be exploited by many applications to make provisions for the bad times. We briefly consider quality adaptation for video playback as an example.

The remainder of the thesis is organized as follows. In Chapter 2, we discuss related work, with an emphasis on TCP-friendly protocols. In Chapter 3, we describe in detail the TEAR protocol as proposed in [24]. In Chapter 4, we present and analyze the results of

our experiments of testing TEAR over the Internet. Chapter 5 is a brief introduction to data transmission over wireless networks. In Chapter 6, we describe how TEAR works over wireless networks and a simple modification we make to the protocol. In Chapter 7, we present and analyze the results of our experiments of running TEAR over wireless networks. Finally, Chapter 8 presents concluding remarks and briefly comments on future work.

Chapter 2

Related Work

We take a look at a few of the different classes of TCP-friendly flow control protocols

2.1 AIMD Flow Control

There are sender-based flow control protocols that use additive increase, multiplicative decrease (AIMD) rate control at the sender similar to that used in TCP. The entire class of such AIMD algorithms are provably stable and fair under steady state. Most of these protocols require the receiver to send acknowledgments for every received packet to detect network congestion. [9] maintains a congestion window which is used directly to control the transmission of new packets. This is in contrast to the indirect use of the congestion window for rate control in TEAR. A sub-class of unicast congestion control mechanisms that are one step removed from TCP are those that use AIMD but do not use a congestion window. The Rate Adaptation Protocol (RAP) [22] uses acknowledgments from the receiver to detect packet losses and estimate RTT and performs AIMD rate control. Instead of using a congestion window, the protocol uses long-term and short-term averages of the RTT to estimate the sending rate for every packet. One characteristic of such pure AIMD protocols is that they do not take into account the impact of retransmission timeouts. Another AIMD protocol proposed in [27] does not use regular acknowledgments, and instead depends on RTP [26] reports from the receiver to estimate the loss rates and round-trip times and do flow control.

2.2 Equation-based Flow Control

Equation-based congestion control was first proposed in [14]. It uses a control equation that gives the maximum acceptable sending rate as a function of the recent loss event rate. In response to the feedback from the receiver, the sender adapts its sending rate according to this control equation. TCP has a response function which characterizes TCP's steady-state sending rate as a function of the round-trip time and the steady-state loss event rate [6]. This is the appropriate control equation that is used for equation-based congestion control algorithms that are targeted to compete with TCP flows on the best-effort Internet. [14] and [17] use different forms of a TCP-friendly control equation to control their rates.

[30] details a mechanism which illustrates the coupling between congestion control and error-resilient scalable video compression. It describes a simple equation-based congestion control mechanism for unicast video transmission over the Internet. The receiver measures the RTT and the loss rate over a fixed multiple of RTT. The sender controls the sending rate and the output rate of an MPEG encoder based on this information and the TCP response function from [14].

TCP-Friendly Rate Control Protocol (TFRCP) [18] uses an equation-based congestion control mechanism for unicast traffic where the receiver acknowledges every packet. The sender estimates the packet loss rate at fixed time intervals and uses this information to update its sending rate according to the TCP response function defined in [17]. There are problems with adjusting the sending rates only at fixed time intervals. It makes the transient response of the protocol poor for fine-grained timescales and it also makes it vulnerable to changes in the RTT and the sending rate.

[6] describes the TCP-Friendly Rate Control protocol (TFRC), which uses the response function in [18] to explicitly adjust its sending rate as a function of the measured rate of loss events, where a *loss event* consists of one or more packets dropped within a single round-trip time.

[25] studies some of the issues in such model-based flow control protocols. It finds that under certain circumstances, the protocols do not converge to the fair bandwidth share, resulting in either over-allocation or under-allocation of bandwidth. This happens because of inaccuracy in estimating the loss rates and from the properties of the TCP-friendly formula that is used. The modeling assumptions of such protocols might not be universally valid under all network environments. For instance, the TCP formula is not reliable when

packet losses or RTT is correlated to the transmission rate of the flow being controlled [18]. Such situations are common in asymmetric networks or under low statistical multiplexing environments.

2.3 Multicast Flow Control

Receiver-driven Layered Multicast [15] was the first scalable flow control protocol that combined layered encoding with receiver-driven flow control. In RLM, receivers join a new layer if they do not observer packet losses for some period of time, and leave a layer when the network is experiencing congestion. *Join experiments*, i.e., the process of joining a new layer under good network conditions, allows receivers to probe for spare bandwidth. One problem with RLM is that the rate increases and rate decreases are both linear, since both of them involve the adding or dropping of a layer. This symmetry in the response to improved network state and deteriorating network state can cause the protocol to be unfair to TCP and it could result in a complete locking out of TCP.

[32] proposes a technique that improves upon RLM by adding TCP-like congestion control techniques on the top of layered flow control. It mimics TCP's AIMD flow control by setting the data rate of each layer exponentially. When a receiver subscribing up to the i-th layer does not see any packet loss for a given time t_i, it adds a layer. The data rate of this new layer and the time are both exponential functions of i . The protocol also adds sender-oriented coordination for join experiments which result in a substantial improvement in max-min fairness of the protocol. The AIMD control improves fairness and stability among the flows belonging to the protocol, but there is no bound on the fairness to TCP flows.

[21] proposes a window-based hierarchical flow control reliable multicast. [7] proposes a similar mechanism. The protocols compute a TCP-window and report these window sizes to the sender. They use a logical tree structure imposed on the receiver set. There is no explicit RTT estimation required because every receiver sends a feedback or a feedback summary. Such window-based approaches are susceptible to the drop-to-zero problem [33][5], where transient network congestion conditions can cause the receiver windows to be reduced. There can be severe under-utilization of the available bandwidth on the bottleneck path because the sender takes the minimum reported window size from the receivers to determine the number of packets to send.

[34] proposes using small equal bandwidth layers called *ThinStreams* to achieve a smooth fluctuation in the rate and allows flow control based on layering to perform fine-grained rate adjustment.

There have been equation-based flow control approaches for multicast flow control. Examples of such model-based protocols include [11], [29] and [30], which propose to use a TCP-friendly formula to determine whether to add or drop a layer. [11] has a scheme which ensures inter-session fairness among competing flows by altering the sensitivity of each layer to respond to the loss rates that are observed.

Chapter 3

The TEAR Protocol

3.1 Brief Overview

TCP Emulation At Receivers [24] essentially models TCP's window adjustment algorithms at the receivers by observing packet arrival events. The receiver maintains a window that is analogous to TCP's congestion window, *cwnd*. Packet losses control the size of this window and the receiver calculates the estimated TCP throughput by using the current congestion window size and the current observed Round Trip Time (RTT). It then smoothes out drastic variations in the calculated rates by taking a long-term exponentially weighted moving average (EWMA) of many such rates. This smoothed estimated rate is typically sent back to the sender (in unicast flows) or could be used to perform receiver-based flow control (layer dropping in multicast flows, for instance).

Here we describe the basic working of the TEAR protocol as proposed in [24].

3.2 The TCP Connection

The TEAR receiver is a finite state machine with states that reflect those of TCP's. There are parallels in TEAR for TCP's slow-start, congestion avoidance, fast recovery and timeout mechanisms. There are a few important differences from TCP which are worth mentioning here:

- In TCP, the reliability part of the protocol is closely coupled with the congestion control part. For instance, if there is an unacknowledged packet in a congestion window of outstanding packets, transmission cannot proceed until this packet has

been reliably transmitted. Since TEAR is not designed for applications that require reliable data transfer - indeed applications that benefit from TEAR, like streaming multimedia, are necessarily designed to tolerate packet losses because of their real-time constraints - any error recovery scheme must be built on top of the flow control mechanism that TEAR provides.

- It needs to be emphasized that TEAR only 'emulates' the various states of the TCP state machine at the receiver. For example, when TEAR enters the FAST-RECOVERY state, it only models the congestion window adjustments of TCP and does not make any real attempts at recovering packets, leave alone recovering them fast. The state names can thus be a little misleading - they are only meant to indicate the nature of the window adjustment TEAR is performing and the corresponding state in the TCP state machine TEAR is trying to emulate.

- Although TEAR uses a congestion window, it is in fact a rate-based congestion control algorithm, i.e., the sender changes its actual sending rate in response to the rate reports it receives from the receiver. This is in contrast to the window-based congestion control algorithm used in TCP, where there is no concept of a sending rate. The congestion window is the parameter that links TEAR closely to TCP and is the key towards achieving TCP friendliness. But this window is used only indirectly and there is a translation from the window-based congestion control that TEAR does during emulation to the rate-based congestion control that is used to modulate the sending rates. Also, it is important to note that TEAR calculates a TCP-friendly rate and the actual TEAR sending rate will, as a rule, be different from TCP's sending rate.

- Unlike TCP, a feedback channel is not essential for TEAR. In unicast sessions, TEAR can use the feedback channel to inform the sender about the current estimated TEAR rates. This feedback frequency is much lower than the per-packet acknowledgment overhead incurred in TCP and makes TEAR a good alternative in situations where the feedback channel is limited in capacity, such as wireless and ADSL networks. In multicast sessions, feedback can be gotten rid off altogether and the receiver can perform layer-based flow control. This mitigates the ACK implosion problem that might be encountered in other sender-based multicast congestion control protocols.

- TEAR models the window adjustment algorithms in terms of rounds instead of round-trip times. A *round* is defined as an interval that contains roughly an arrival of *cwnd* number of packets. Unlike TCP, where reception of acknowledgments determine the round or RTT, TEAR can recognize the end of a round at the receiver, while receiving packets.

- TEAR uses the concept of an *epoch* to calculate a smoothed TCP friendly rate. The idea of an epoch is completely absent in TCP. The TEAR receiver keeps a history of the estimated per-round rates and sets the TEAR transmission rate to an averaged rate over some long-term period - a period that encompasses a few epochs. Loosely, an epoch can be defined as a period that begins when the receivers enters the SLOW-START or the CONGESTION-AVOIDANCE state.

3.3 Assumption of Rate Independence

The problem TEAR faces is to estimate the throughput of a TCP connection over the same end-to-end path only by observing packet arrival events at the receiver. As mentioned before, the transmission rates of TEAR and TCP, as a rule, will be different. The question then arises: Does a window of x packets in a TEAR connection have the same loss probability as that in the *virtual* TCP connection it is emulating? In other words, how does the difference in transmission rates affect packet loss probabilities?

In TEAR, it is assumed that the probability of having a *packet loss event* within a window of x consecutively transmitted packets does not depend on the transmission rate. That is, no matter how large or small the intervals they are transmitted in, the probability that at least one packet in that window is lost is the same given that the network conditions do not change during the transmission period. This assumption is called *rate independence*.

There is a need to justify the assumption of rate independence, because it plays an important role in the overall functioning of TEAR. Rate independence holds if packet losses happen independently and are not correlated. But in the Internet, where drop-tail routers dominate, packet losses are indeed correlated and if a packet is dropped at a routers, the likelihood of the next packet being dropped increases with the proximity in real-time of the second packet to the first. In such conditions, rate independence does not hold.

A loss event can informally be defined to be a single loss burst, or losses within

the same TCP congestion window. TEAR is interested in *packet loss events* and not in individual packet losses, because TCP reacts only once per loss event. Loss bursts on the Internet are typically short and the loss correlation interval does not span long - typically such intervals last less than one RTT. It is common to assume in TCP that loss events are not correlated and happen independently [14][27][29][18][30][16]. Thus, when emulating TCP, TEAR ignores losses that are likely to be correlated and bunches them together as a single loss event. Under such operating conditions, it is believed that rate independence can generally be assumed.

3.4 TEAR Rounds

TCP maintains a congestion window, *cwnd*, which indicates the number of packets in transit from the sender to the receiver, i.e., the number of unacknowledged packets outstanding in the "bandwidth pipe". The congestion window moves when TCP's "ack clock", which is driven by acknowledgments, ticks. TEAR maintains a variable analogous to the TCP congestion window at the receiver, instead of the sender. The window is updated according to the same algorithms based on the various events that the receiver observes - packet arrivals, packet losses and timeouts.

A TEAR round is defined as an interval that contains the arrival of roughly *cwnd* number of packets. The TEAR transmission session consists of a succession of non-overlapping rounds. The idea of a round is essential because TEAR might be sending at a different rate from the emulated TCP, so the window update functions cannot be described in terms of real-time, for instance, the round-trip time, as it is done in TCP.

In TCP, a round finishes when all the packets in the congestion window are acknowledged, i.e., rounds are synonymous with the round-trip time. In TEAR, the receiver recognizes the end of a round when it simply receives a congestion window full of packets. The duration of a round depends on the inter-arrival times between the packets, which in turn depends on the transmission rate. To account for the mismatch between the TEAR and TCP rounds, TEAR associates a fictitious RTT with each round. When calculating the transmission rate for one round, TEAR divides the congestion window size by the current estimate of the RTT, instead of using the real-time duration of the round.

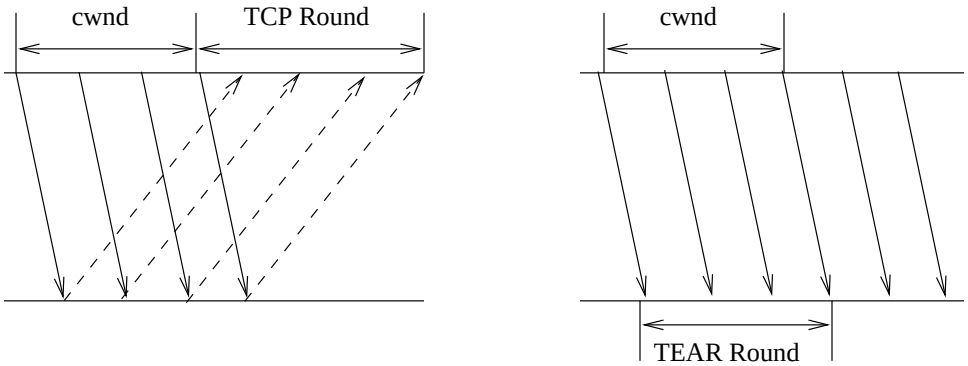

Figure 3.1: Illustrating the difference between TCP and TEAR rounds

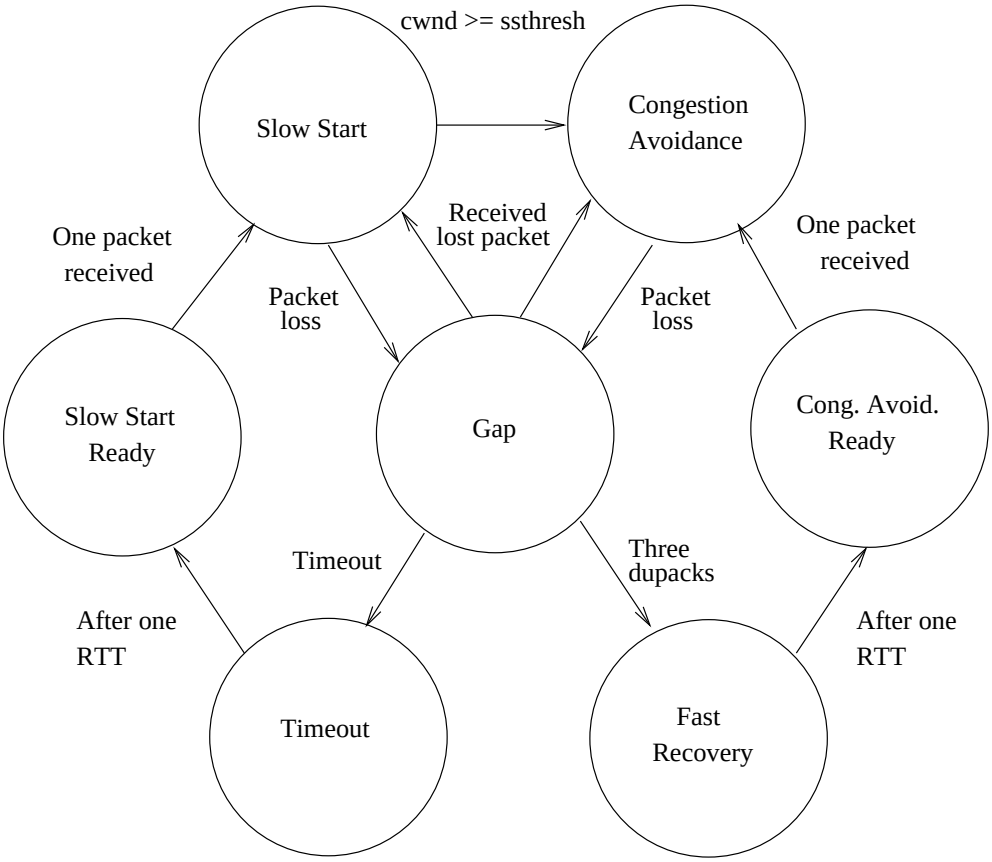

Figure 3.2: The TEAR State Transition Diagram

3.5 State Transition in TEAR

The TEAR receiver can have seven possible states: *slow-start-ready, slow-start, congestion-avoidance-ready, congestion-avoidance, fast-recovery, timeout* and *gap*. As the names indicate, SLOW-START, CONGESTION-AVOIDANCE, FAST-RECOVERY and TIMEOUT correspond to the states of TCP during slow-start, congestion avoidance, fast recovery and timeout. The other three states are intermediate states in TEAR that are required for the receiver to restart after a state transition occurs.

TEAR maintains three variables for its operation. *cwnd* is the value of the current congestion window and *lastCwnd* is the value of the congestion window at the end of the previous round. A round ends when *lastCwnd* number of packets are received from the time the round began. A new round also begins whenever the state changes to FAST-RECOVERY-READY or SLOW-START-READY. No new round starts in the GAP, FAST-RECOVERY or TIMEOUT states. The variable *ssthresh* parallels the slow-start threshold used in TCP and is used to determine when TEAR should move from the SLOW-START mode to the CONGESTION-AVOIDANCE mode of window growth.

3.6 TEAR Window Increase Algorithm

3.6.1 Slow Start

The transmission session begins with the state being initialized to SLOW-START-READY. *cwnd* is set to 1 and *ssthresh* is set to a default value larger than 2. The reception of the first packet sets the TEAR state machine in motion and the state changes to SLOW-START. In slow-start, TEAR tries to emulate TCP's slow-start behavior. For each in-sequence reception of a packet, the congestion window is incremented by one. Every packet has a unique sequence number which identifies it and TEAR uses this to detect packet losses and in-sequence receptions. Slow-start results in an exponential growth of the congestion window in every round. As has been observed in TCP's context, this does not really justify the term *slow* start. At the beginning of each round, *lastCwnd* is updated to the value of *cwnd* to be used in computing the next round's duration. When *cwnd* increases beyond the slow-start threshold, the state changes to CONGESTION-AVOIDANCE.

3.6.2 Congestion Avoidance

Congestion avoidance is the state in which TCP linearly probes for available bandwidth. TEAR increments the congestion window by $1/lastCwnd$ for each in-sequence reception of a packet. This emulates TCP's congestion avoidance behavior.

3.7 TEAR Window Decrease Algorithm

Whenever a packet loss is detected (the sequence number received was greater than the expected sequence number), TEAR changes its state from SLOW-START or CONGESTION-AVOIDANCE to GAP (the name indicating that a gap was detected in the packet sequence). In TCP, when a packet loss occurs, either the sender does not receive any acknowledgments (in which case it times out) or it keeps receiving duplicate acknowledgments (in which case it does fast-recovery when the dupack threshold is reached). TEAR tries to emulate this behavior of TCP in the GAP state.

In the discussion below, we assume that the last sequence number received was n and packet $n + 1$ was lost.

3.7.1 Fast Recovery

In TCP, because acknowledgments are cumulative, packets that are received after a packet loss trigger duplicate acknowledgments. In order to distinguish between packet losses and packet reordering, TCP has a "dupack" threshold, which is typically set to three. That is, if the sender receives three acknowledgments for a packet (two "dupacks") before it times out, it enters the fast recovery phase, assumes that the packet was lost and retransmits the lost packet. If TCP finishes sending a congestion window full of data and this window contains the lost packet, the sender cannot proceed and the congestion window cannot move until the lost packet is acknowledged. This means that in TCP, at most *lastCwnd* - 1 packets can be sent after the retransmission of the lost packet. TCP reduces its window only once for all the losses of packets transmitted within the same congestion window. If the lost packet remains unacknowledged and the retransmission timer goes off, TCP times out, retransmits the packet and enters slow-start.

To emulate the dupack behavior of TCP, the TEAR receiver enters the FAST-RECOVERY state from GAP when at least two packets are received. These two packets

would have triggered duplicate acknowledgments in TCP, and this in turn would have triggered the fast recovery mechanism. In addition, these two packets must fall within a window of length *lastCwnd* from the lost packet. The latter condition emulates TCP's characteristic that it would have transmitted at most *lastCwnd* - 1 packets after the lost packet. A third condition is that these packets must be received within a period $T_timeout$ after the reception of last in-sequence packet. $T_{timeout}$ is an estimated time for *lastCwnd* packets to arrive.

It is possible that there is packet reordering in the network. In this case, the lost packet $n + 1$ will be received and the receiver returns to the last state before GAP and updates cwnd according to the window increase algorithm. All the packets received before the reception of the reordered packet and have a higher sequence numbers are considered to be received at once and *cwnd* is incremented for each of those packets. This makes the receiver behave as if there were no packet losses.

Since TCP responds only once for all the losses within the same congestion window, TEAR ignores all the packets that are received for one RTT period in the FAST-RECOVERY state. There are two ways of implementing this waiting - either the receiver could set off an RTT timer or it could send a feedback packet to the sender and wait for an ack from the sender (this would give an accurate estimate of the current RTT). We adopt the former approach of setting a timer, since this way the entire process can be localized at the receiver.

After ignoring packets for an RTT period, the receiver goes back to the state it was in before entering GAP. If it was in CONGESTION-AVOIDANCE, the receiver goes back to CONGESTION-AVOIDANCE-READY. In this case, the receiver reduces *cwnd* and *lastCwnd* to half of the value of cwnd at that time. The receiver moves to CONGESTION-AVOIDANCE at the reception of a new packet and thereafter *cwnd* is incremented according to the window increase algorithm.

3.7.2 Timeout

In TCP, when a packet is not acknowledged until its retransmission timer expires, i.e., the packet is lost and fast retransmission was not triggered, a timeout occurs. The packet is retransmitted and TCP goes back to slow-start. The retransmission timers are set to a value large enough so that fast retransmission can be triggered within this interval.

TCP, as we observed before, sends at most *lastCwnd* - 1 number of packets after the lost packet. If the TCP receiver does not receive at least three packets from this window, then fast retransmission cannot be triggered and the sender will be forced to timeout.

To emulate this behavior, if the TEAR receiver does not enter FAST-RECOVERY from GAP until $T_{timeout}$ time has elapsed, it enters TIMEOUT. Also, if it does not receive any packets for an interval of length $T_{timeout}$ while it is in CONGESTION-AVOIDANCE, the receiver enters TIMEOUT.

TEAR times out if it is in the GAP state and less than three packets are received within the time interval in which *lastCwnd* - 1 packets are expected. Thus $T_{timeout}$ is the expected time for *lastCwnd* number of packets to arrive. This is calculated from the inter-arrival time of packets as:

$T_{timeout} = T_{interarrival} \times lastCwnd \times 2 \times DEV$

DEV is the deviation in the RTT estimates which is computed in the same way as in TCP by the sender from feedback. When such feedback is not feasible, the deviation can also be calculated by multiplying the deviation of the difference of the sending and receiving timestamps by $\sqrt{2}$.

After entering TIMEOUT, the receiver waits for an RTT period to ignore packets that might belong to the same loss event as the one that caused the timeout. At the end of the RTT period, the state is changed to SLOW-START-READY, *ssthresh* is set to the half of min*cwnd*, 2, *cwnd* and *lastCwnd* are set to 1 and $T_{timeout}$ is doubled. Effectively the system goes back to the initial state.

3.8 Smoothing the estimated rate

TEAR estimates TCP's transmission rate by dividing *cwnd* by the current estimated RTT. However, TEAR cannot set its transmission rate to this value, because that would retain TCP's wide rate fluctuations. To avoid this, TEAR sets the transmission rate to an averaged rate over a longer period.

To do this smoothing of estimated rates, TEAR has the concept of an epoch. An *epoch* is a period that begins when the receiver enters SLOW-START or CONGESTION-AVOIDANCE. When a new epoch starts, the current epoch ends. This means that an epoch period is demarcated by packet losses, since it is when TEAR observes losses that it enters SLOW-START or CONGESTION-AVOIDANCE.

Epoch	k	k - 1	k - 2	k - 3	k - 4	k - 5	k - 6	k - 7
Weight	1/6	1/6	1/6	1/6	2/15	1/10	1/15	1/30

Table 3.1: TEAR weights used for smoothing the estimated rate

At the end of each round, the receiver stores the values of *cwnd* and RTT to a history array. If the round involved a TIMEOUT, then *cwnd* and RTO are stored instead. Suppose the current epoch is the kth epoch. The receiver then calculates the rate sample of epoch k by dividing the sum of all the cwnd samples of the kth epoch by the sum of the RTTs or RTOs recorded in that epoch. At the end of each epoch, the rate is set to the rate sample of that epoch. This results in a much more smoothed rate adjustment.

There might still be some unnecessary rate fluctuations present because some rate samples may not be representative of the actual fair share rate due to noise in loss patterns, which is fairly common in current environments. To filter out this noise, TEAR applies a weighted averaging over rate samples taken over several W epochs in the past. At the end of each round, the receiver computes the weighted average of the last W rate samples taken at the end of each of the last W epochs. If the kth epoch is in progress, its sample is not used until it is large enough to be reliable, or until the epoch ends. Thus, two weighted averages are computed: of epoch k - 1 to k - W - 1 and from k to k - W - 1. The larger of the two averages is used for calculating the feedback rate to the sender.

W is chosen to be 8 and the following weights were used for the Internet tests of TEAR.

Chapter 4

TEAR Over the Internet: Experimental Results

4.1 Experimental Setup

The two main objectives of our Internet experiments for evaluating and verifying the performance of TEAR are the following:

- To ascertain that TEAR is *fair* and *TCP-friendly*, i.e., study how TEAR flows share bandwidth with each other and how they share bandwidth with TCP.
- To study the *stability* of TEAR, i.e., study how TEAR flows react when flows join and leave, whether the protocol eventually reaches the fair and TCP-friendly rate.

We conducted Internet experiments between the United States and South Korea to analyze the performance of TEAR and compare it with that of TCP. These tests were run between two Linux machines running Red Hat Linux 6.2, one stationed in NC State's Centennial Campus and the other stationed in Seoul, South Korea. The server was running in the United States and the clients were run from Korea, i.e., the data transfer was from the US to Korea. The experiments were run through the week of April 3 - April 10, 2001 during various times of the day. In this chapter, we take a look at some of the salient and representative results that were obtained during these runs.

In evaluating and verifying TEAR's performance, we look at the following scenarios.

- Single TCP run - to find out the available bandwidth of the network.
- Single TEAR run - to verify whether TEAR achieves this available bandwidth and compare the smoothness of the rate variations with that of TCP.
- Multiple TCP and multiple TEAR flows together - to observe how TCP and TEAR share the available bandwidth and study the TCP friendliness of TEAR.
- Scenarios where flows join and leave - to observe how TEAR flows respond to dynamic changes in available bandwidth and study the stability of TEAR.

4.2 Finding the Available Bandwidth

Single TCP flow Figure 4.1 shows the instantaneous rate variations and the aggregate throughput of a single TCP flow that was run between the US and Korea. TCP gets a throughput of about 275 kbps (35 KB/s). This is the fair share bandwidth available in the network. We notice that the there are wide fluctuations in the TCP rates because of the packet losses it encounters in the network.

Single TEAR flow To see whether TEAR can achieve this available bandwidth, we ran a single instance of TEAR immediately following the TCP run. Figure 4.2 shows the instantaneous rate variations and aggregate throughput of TEAR and we see that TEAR gets a throughput of around 250 kbps (32 KB/s), which is close to the fair-share bandwidth that TCP achieved under similar conditions. More importantly we can see that the rate fluctuations in TEAR are much smoother than those observed in TCP. Both the magnitude of the rate deviation and the frequency of fluctuation, are lower for TEAR compared with TCP.

4.3 Fairness and TCP friendliness

Fairness amongst TEAR flows Figure 4.4 shows the aggregate throughputs from a run of four TEAR flow sharing the network. The fair-share for four flows would be around 70 kbps. We see that the TEAR flows reach a bandwidth close to this fair share. The four flows achieve throughputs between 62kbps and 70kbps. Thus we find that TEAR flows are

fair to each other. The TEAR rates fluctuate around their fair share and the magnitude of these variations are small.

Fairness amongst TEAR and TCP flows Figures 4.5-4.12 show the instantaneous rate variations and the aggregate throughputs of various runs with multiple TCP flows and multiple TEAR flows sharing the network. The results show that in general, TEAR flows compete fairly with TCP and they show smoother rate fluctuations than TCP. Figure 4.5 shows the results from a run of 1 TEAR and 1 TCP flow running together. The fair share would be around 140 kbps and TCP, expectedly, reaches the fair-share. TEAR achieves a throughput of around 125 kbps, close to the fair-share. Figure 4.8 is from a run with 2 TEAR and 2 TCP flows sharing the network and Figure 4.12 is from a run with 4 TEAR and 4 TCP flows. In all the cases we see that TEAR flows reach the fair-share or get close to it. The throughput of TCP is higher than that achieved by TEAR. This is because, due to rate smoothing, TEAR does not probe the available bandwidth as aggressively as TCP and hence does not make use of good network conditions as well as TCP does. On the other hand, TEAR does not react in a drastic way like TCP when the network is congested and thus is able to make up for the lost bandwidth and hover around the fair share. When the number of TCP flows is increased, the difference between the maximum throughput of TCP flows and the minimum throughput of TEAR flows grows bigger. That is, as the number of competing TCP flows increases, the throughput that TEAR gets tend to fall away from the fair-share. But we find that TEAR still is able to come close to the fair-share of bandwidth.

4.4 Smoothness or Rate Fluctuations

In all the runs, TEAR shows much fewer and lower fluctuations in its instantaneous rates than TCP. As observed before, the cost of rate smoothness is less throughput during the times the network is good (because of less aggressive bandwidth-probing) and a slower response to congestion when conditions are bad (because of less conservative reduction in the sending rate). It is important to note that a drastic reduction in the sending rate is not necessary in order to prevent a congestion collapse. As long as the protocol responds to persistent congestion, i.e., congestion spanning longer time intervals, it does not pose a threat to network stability. This is exactly what TEAR does. It does not cut its rate dramatically as soon as it detects packet losses, but as congestion persists, we can see that

there is a downward slope in TEAR's sending rates which continues until the network gets out of its congested state. When the congestion period is over, TEAR starts increasing its rates again.

4.5 Stability

When a network undergoes some perturbation because of flows joining and leaving, when the network reaches the steady state, a stable protocol must eventually reach the fair and TCP-friendly rate, no matter what the state of the protocol at the end of the perturbation is. Stability is an important criteria in real networks because of the dynamic ways in which network conditions keep changing. To study the stability of TEAR, we ran TEAR flows and TCP flows together with unequal flow lengths, i.e., flows joining or leaving at different times. We discuss one such run here. Figures 4.13 and 4.14 show the instantaneous rate and aggregate throughputs of 4 TEAR and 4 TCP flows together. The TCP flows were all stopped after 75 seconds. The fair-share bandwidth for the first 75 seconds is about 35 kbps and we see that the TEAR flows get a throughput close to this fair-share. After 75 seconds, the TCP flows leave and this makes available twice the bandwidth for each TEAR flow. From figure 4.13, which shows a representative TEAR flow's instantaneous rate, we see that the TEAR flow probes for this extra available bandwidth and eventually reaches the new fair-share. Figure 4.14 shows how the throughput of each of the TEAR flows keeps increasing and converges towards the fair-share. TEAR starts reacting to new favorable conditions pretty soon, but because of its less aggressive bandwidth probing, it takes some time to reach its steady state bandwidth. But it does eventually attain the fair and TCP-friendly rate and is stable with respect to changing network conditions.

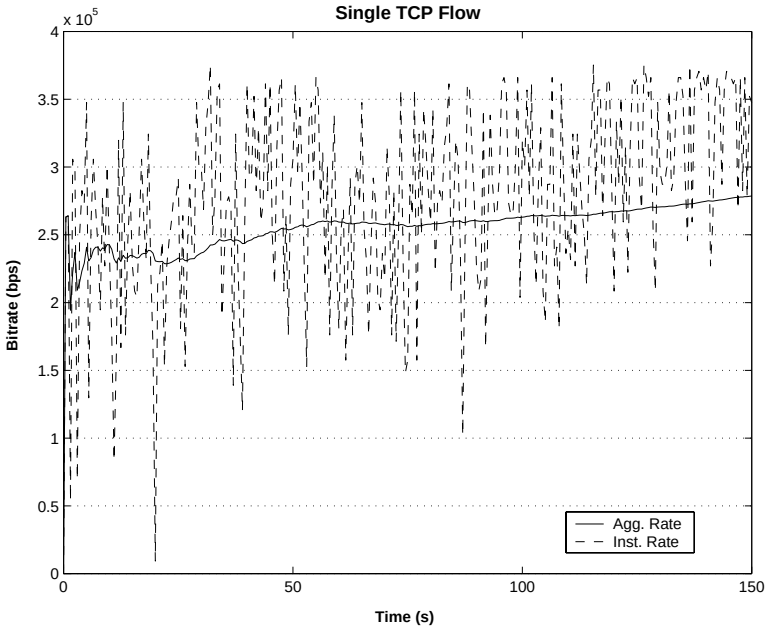

Figure 4.1: Instantaneous rate and aggregate throughput of a single TCP flow

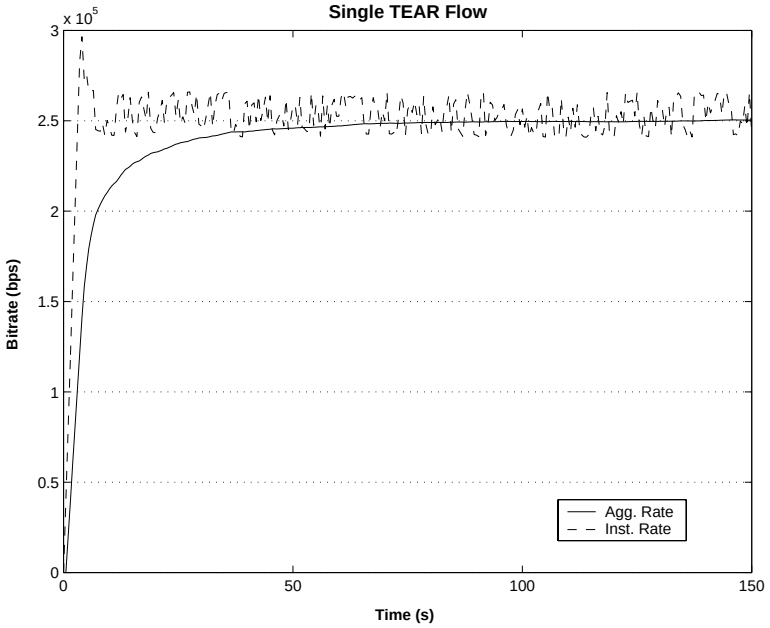

Figure 4.2: Instantaneous rate and aggregate throughput of a single TEAR flow

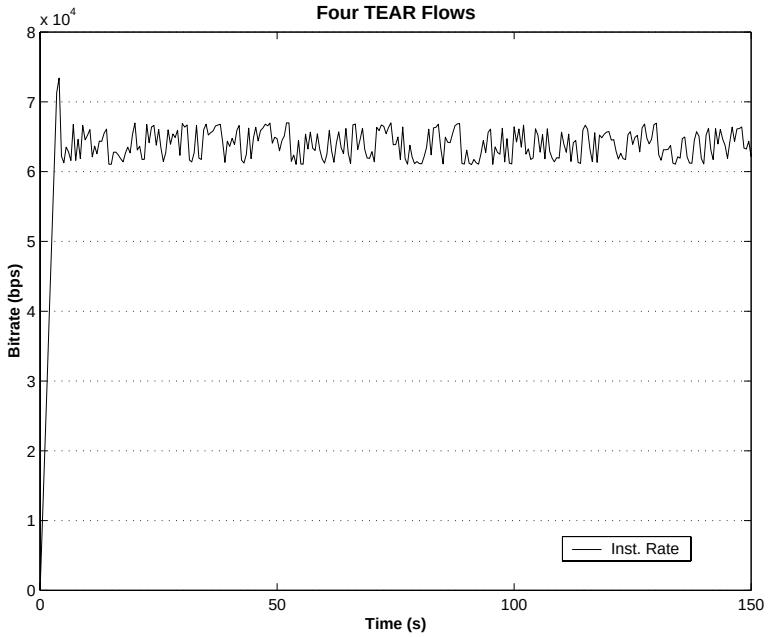

Figure 4.3: Instantaneous rate of a TEAR flow sharing the network with 4 TEAR flows

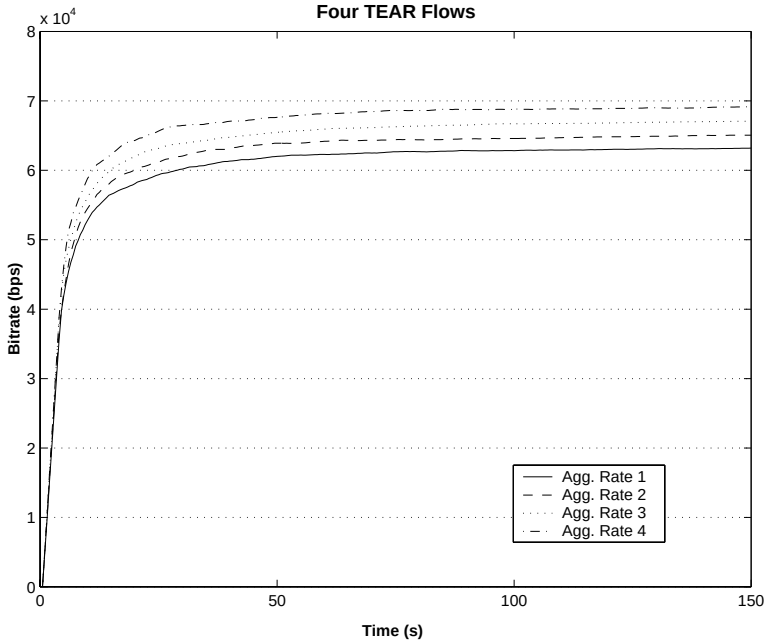

Figure 4.4: Aggregate throughputs of four TEAR flows running simultaneously

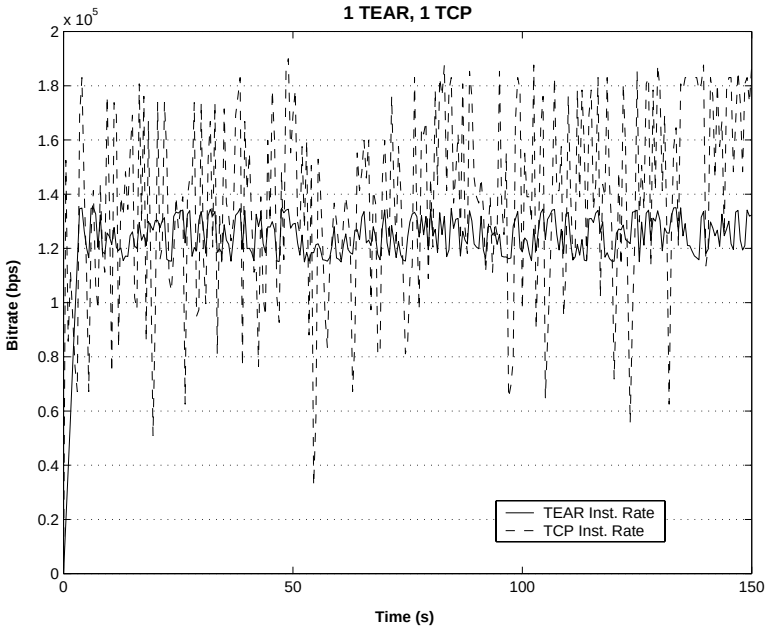

Figure 4.5: Instantaneous rate of 1 TEAR flow and 1 TCP flow sharing the network

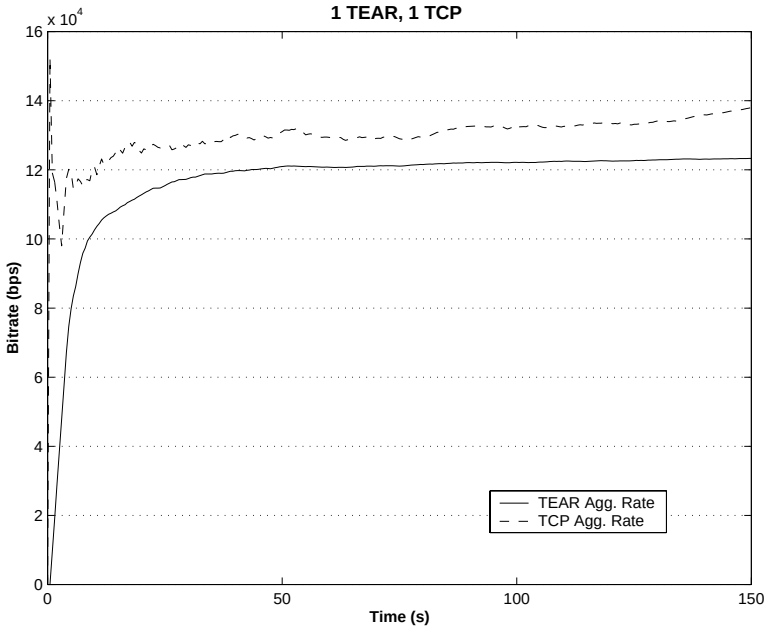

Figure 4.6: Aggregate throughputs of 1 TEAR flow and 1 TCP flow sharing the network

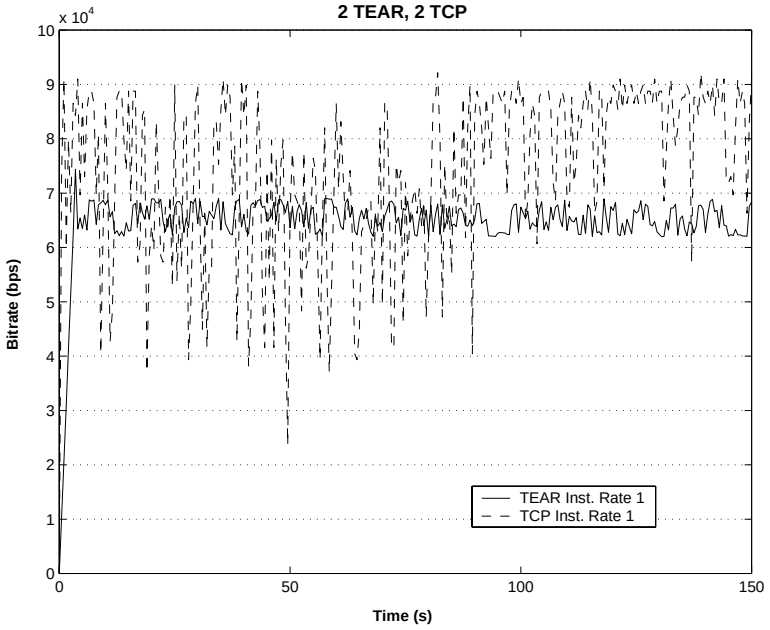

Figure 4.7: Instantaneous rates of representative TEAR and TCP flows

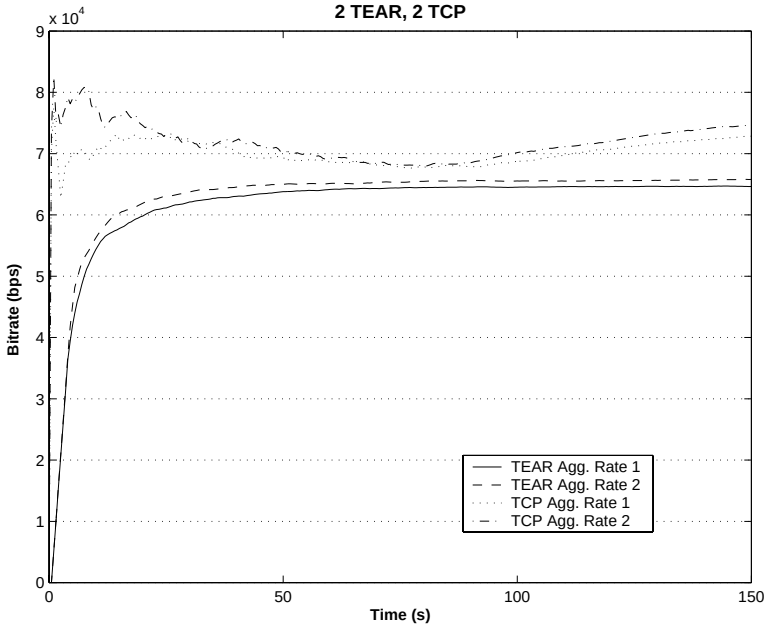

Figure 4.8: Aggregate throughputs of 2 TEAR flows and 2 TCP flow sharing the network

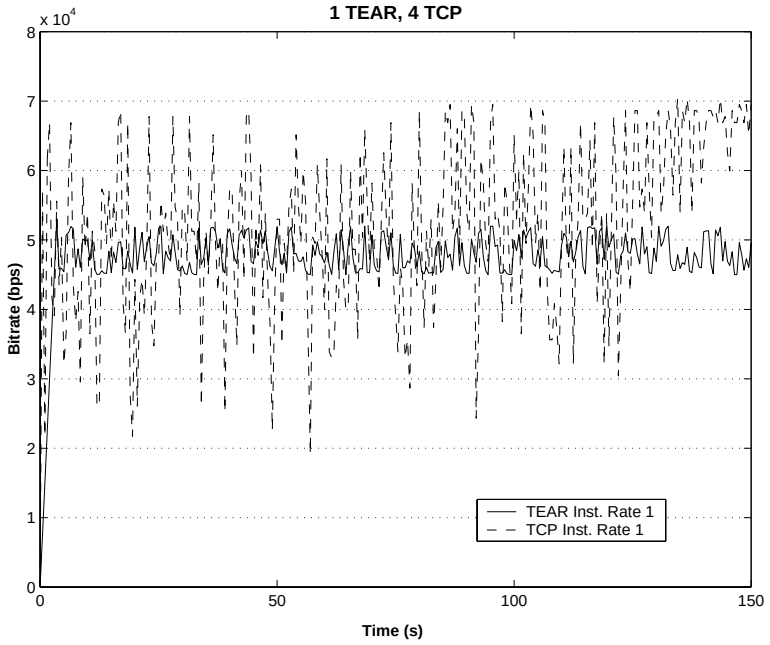

Figure 4.9: Instantaneous rates of representative TEAR and TCP flows

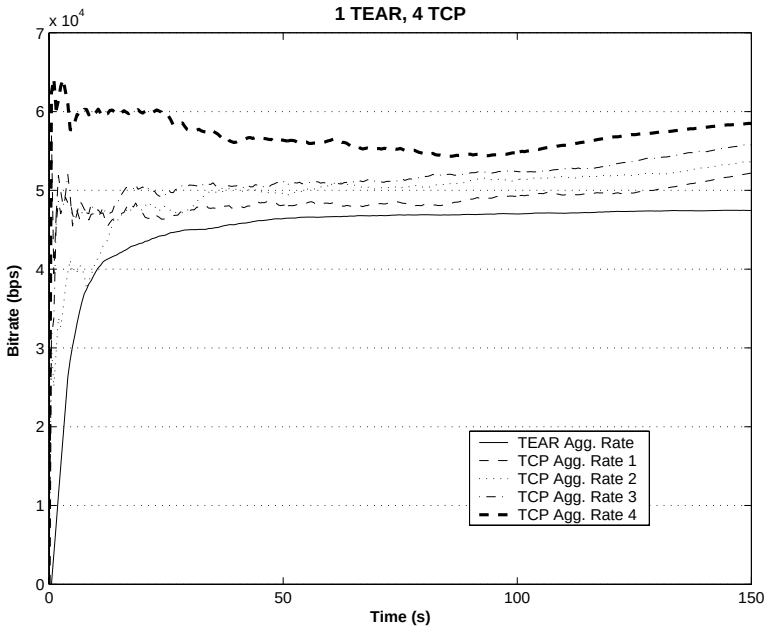

Figure 4.10: Aggregate throughputs of 1 TEAR and 4 TCP flows sharing the network

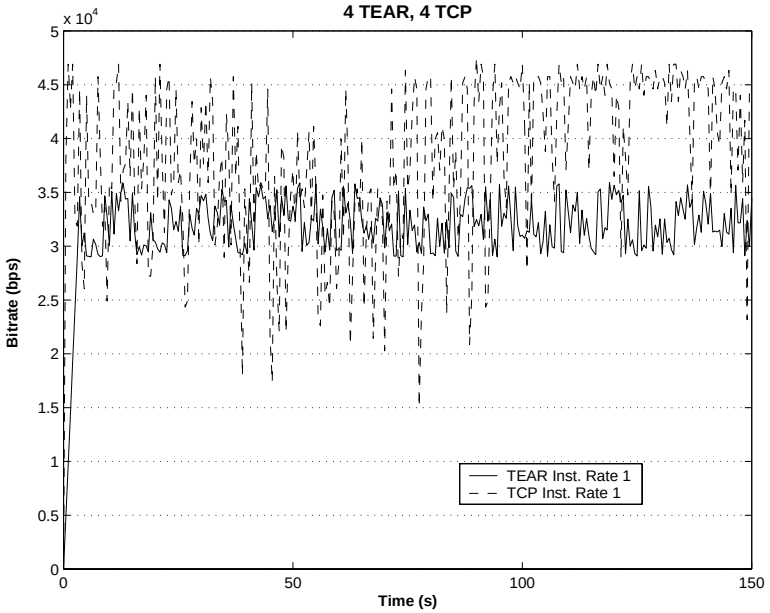

Figure 4.11: Instantaneous rates of representative TEAR and TCP flows

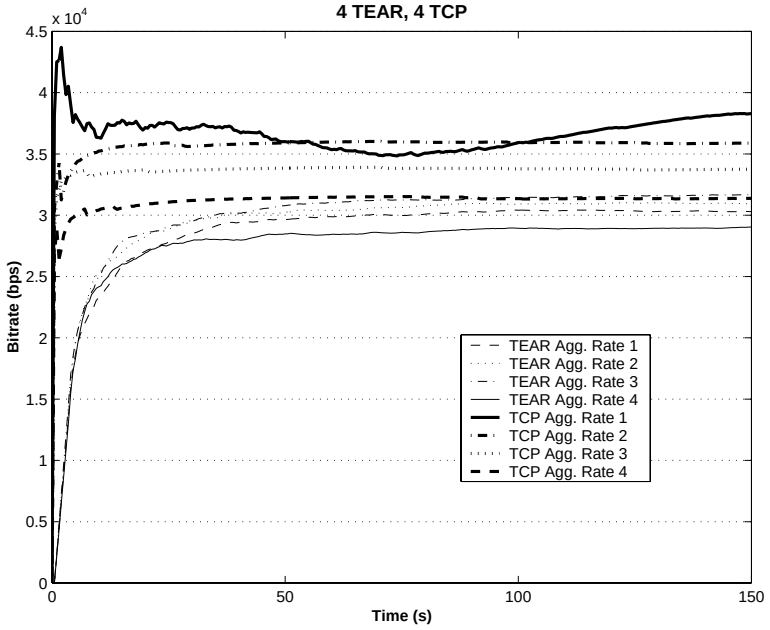

Figure 4.12: Aggregate throughputs of 4 TEAR and 4 TCP flows sharing the network

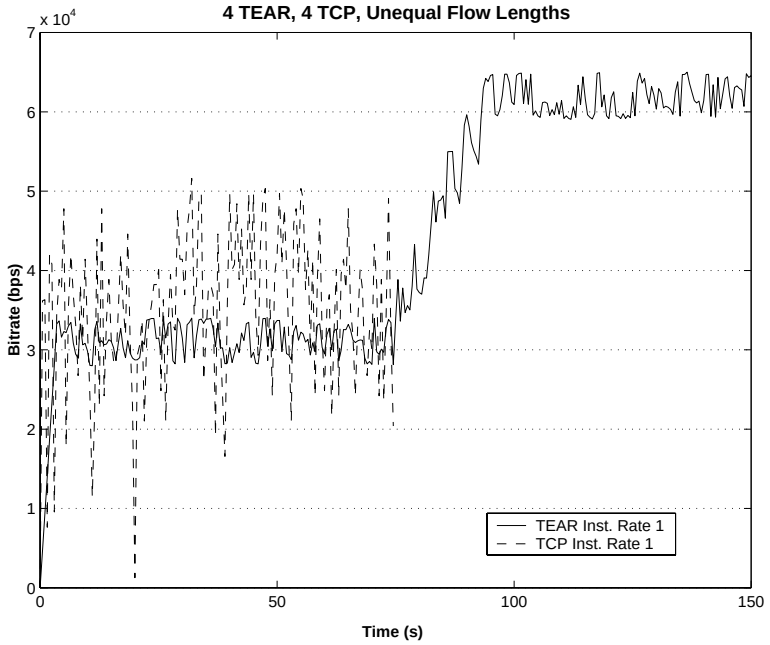

Figure 4.13: Instantaneous rates of representative TEAR and TCP flows

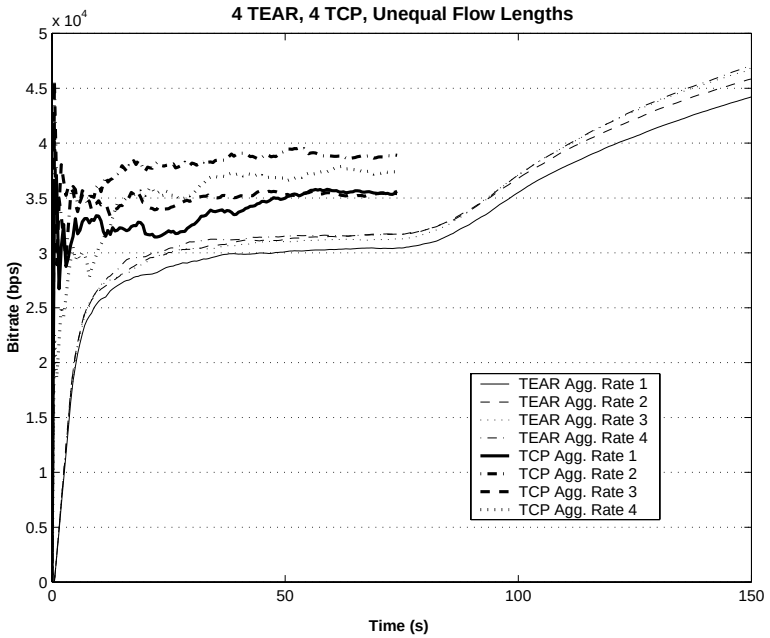

Figure 4.14: Aggregate throughputs of 4 TEAR and 4 TCP flow sharing the network, the TCP flows are stopped halfway through the run

Chapter 5

Data Transmission Over Wireless Networks

5.1 Wireless Networks

The use of radio waves for communication has been known for over a century now due to the pioneering works by inventors like Marconi, De Forest, and Armstrong. Starting from Marconi's demonstration of Trans-Atlantic radio communication in 1901, the wireless communication field has grown by leaps and bounds to the current state-of-the-art.

The last decade has seen enormous advances being made in the hardware technology for wireless communication, which has enabled portable and affordable wireless devices to be mass produced. Coupled with the rapid expansion of the Internet, this has resulted in the commercial availability of wireless technologies.

Many wireless and mobile networks are organized in a *cellular topology*. A typical topology with mobile users is implemented using a wired, packet-switched backbone network and a wireless network. The wireless network itself is organized into geographically defined cells. Some of the fixed wired hosts are augmented with a wireless interface and these are called the *base stations*. The base stations provide a control point between the wired network and the wireless network and route packets between the two. A fixed host in the wired network communicates with a mobile host in the wireless network through the base station of the geographical cell the mobile host is present in. When the mobile user moves between cells, the task of forwarding information from the wired network to the user is

transferred to the base station in the new cell the mobile user occupies. The user's location information is updated in the wired network and this process is called a *handoff*.

5.2 Packet Losses in Wireless Networks

Packets on the Internet may get lost either due to congestion or due to packet corruption in the underlying medium. Wired links typically have extremely low bit error rates, so it is safe to assume that almost all packet losses are due to congestion in the network. TCP's reaction to losses is based on this assumption.

This assumption is not true on wireless networks. Errors on wireless links tend to be frequent and bursty and occur due to a variety of reasons such as *path loss*, *shadow fading*, *multipath fading*, *noise* and *interference*.

Path loss is defined as the ratio of signal power at the receiver to that at the sender and is a strong function of the distance between the transmitter and the receiver [20]. Typically for free space the received power varies inversely with the square of the distance of separation, i.e., the path loss exponent is 2. For typical outdoor environments, the received power falls even more drastically with distance, with path loss exponents varying between 3 and 5. The higher the path loss exponent, the higher the bit-error rate for the same transmitter power.

Shadow fading occurs because of objects and obstructions in the path of the transmitted signal. *Multipath fading* occurs because of multiple versions of the same signal interfering at the receivers as a result of the original signal being reflected from various obstacles on the way. This results in a degradation of the signal because destructive interference is the more likely to occur than constructive interference.

It is also possible that users in geographically close cells use the same frequency for communication, because of *frequency reuse* that is characteristic of cellular networks. This can result in *co-channel interference*.

All these pathologies cause a great deal of bit errors in wireless networks. Studies have shown that average bit-error rates vary between 10^{-2} and 10^{-3} [2] with varying conditions on wireless networks. Thus TCP encounters packet losses that are unrelated to congestion. Since it has no way of distinguishing such losses from congestion-induced losses, these losses also trigger the congestion control mechanisms of TCP, reducing the congestion window by half. The bursty nature of wireless losses also causes TCP to timeout more often

than in wired networks. All these together lead to unacceptably low TCP throughput and under-utilization of the available bandwidth.

A wide range of wireless technologies exist today, which differ widely in their characteristics. Originally, the term wireless simply meant "without wires" and this property could be achieved using radio waves, infrared or other means for propagation. In this thesis, when we use the term wireless, we specifically are referring to communication using radio waves.

Existing wireless networks can be classified broadly into *Wireless LAN*s, *satellite link*s, and *Wireless Wide Area Network*s. These networks share some characteristics, but they also have distinct properties that sometimes it is better to treat them as different environments for communication. In this thesis, we are concerned only with Wireless Wide Area Networks, which are cellular systems that can be used for data communication.

5.3 Various Approaches To Handle Error Prone Wireless Links

The way TCP assumes that losses on a wireless link as congestion losses has made enhancements of TCP performance necessary for wireless networks. This has resulted in a variety of approaches towards finding a solution to the problem of TCP underperformance.

An early attempt classified the existing solutions into three different categories: end-to-end proposals, split TCP, and link layer proposals [2]. Another way of classification is offered by [31]. In general, all methods either attempt to hide the error losses from the sender or make the sender know the cause of the loss. The first approach aims at ideal network behavior, i.e., the network recovers from the losses transparently to TCP. The second approach aims at ideal TCP behavior, i.e., TCP simply retransmits the corrupted packets without invoking its congestion control mechanisms. On a functional level, the methods are grouped into the following categories: *link-level mechanisms*, *split connection approach*, *TCP-aware link layer*, *TCP-unaware approximation of TCP-aware link layer*, *explicit notification*, *receiver-based discrimination* and *sender-based discrimination*.

[12] classifies the enhancement methods into: *pure end-to-end*, *transport-layer splitting*, *transport-layer caching*, *cross-layer signaling* and *pure link-layer methods*.

A *pure end-to-end* method relies on making enhancements to TCP at the end

hosts, without involving any intermediate hosts. This retains the end-to-end semantics of TCP and the standard TCP mechanisms like slow-start and congestion avoidance. Instead improvements are made to parameters like the window size, the initial window and schemes like selective acknowledgments and explicit congestion notification are added.

In the *transport layer splitting* approach the wired and wireless portions of the link are handled separately The TCP connection from the fixed host ends at a performance Enhancing proxy (PEP) and the data is delivered to the mobile using a special data delivery protocol. Even though that is a great advantage to this approach , it violates the semantics of a TCP connection because of the faked ACKs sent by the PEP.

The *transport layer caching* approach eliminates the problem of maintaining the hard state in the proxy that is present in the previous approach. The loss of the soft state on the proxy can affect the performance, but does not prevent the end-to-end data delivery by TCP. The best-known implementation of the soft-state proxy concept is the Snoop protocol [4]. Snoop examines packets in PEP in a way that allows to detect TCP segment losses and recover locally by retransmitting the cached segment. The main shortcoming of Snoop is low performance in the presence of a high level of congestion losses [13].

Another approach is to handle it completely at the link layer trying to isolate the local problems of the wireless link from the higher layers. The wireless links can use Automatic Repeat Request to recover from lost packets by using link-level retransmissions [28]. Here the link errors are not visible to the upper layers, but at the expense of variable delays in the data delivery. Some link-layer protocols provide semi-reliable data delivery, by performing only a small number of local retransmissions before discarding a packet. The current research favors highly persistent link-layer recovery [13].

But in cases like real time video transmission the link layer approach may not work satisfactorily and sometimes even be harmful to the application because data needs to be delivered in time or not at all To alleviate this problem the concept of flow-adaptive links was proposed where the required Quality of Service (QoS) can be specified for a specific packet. The QoS requirement is given to the link layer in the type of service field in the IP header.

Chapter 6

TEAR Over Wireless Networks

6.1 Delay-Based Congestion Control For Wireless Networks

The degradation of TCP's performance on wireless networks is mainly due to the inability of the protocol to distinguish between losses caused by congestion and losses on the unreliable wireless link. Bandwidth-hungry applications with real-time constraints such as video-on-demand and group services are beginning to be commercially available in various wireless markets - especially in Europe and Asia. It is necessary that these applications control their sending rates, so as to avoid network collapse in the packet-switched network they share with other applications. At the same time they should avoid under-utilization of the network by not responding to false signals that bring about a reduction in their sending rates . Thus we have two contrasting requirements - not overshooting the available bandwidth and not underestimating the capacity either. To achieve this, instead of using ambiguous packet losses as an indication of network congestion, it is worth considering an alternative parameter - end-to-end delay - to detect congestion.

When packets are pumped out into the network at a rate higher than that can be supported by the network at the time, they get queued in router buffers. As the load on the network increases and queues build up, the round-trip delay increases and this information can be used by senders to adjust their sending rates. Delay-based congestion control for wired networks has been proposed before [8], which uses the gradient of the delay-window curve to derive an expression for the optimal window size to be used. Such schemes have been proven to be fair, convergent and adaptive to network changes, but the literature has been dominated by packet-loss based congestion control, one reason being that TCP, the

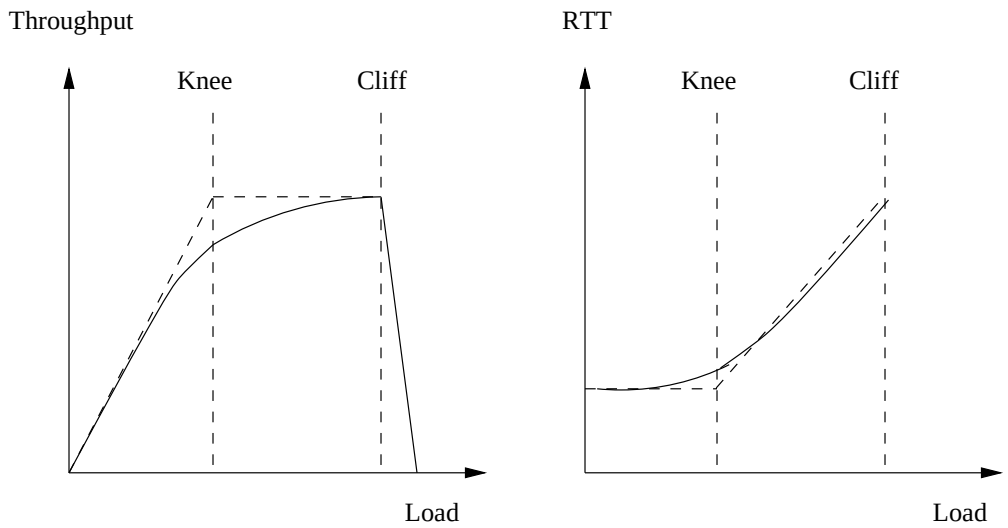

Figure 6.1: Variation of throughput and RTT with network load [8]

dominant protocol of the Internet, uses packet losses to detect congestion. Delay-based schemes thus have not been deployed in real networks and have remained an academic interest.

For our experiments with wireless networks, we have kept the window and rate smoothing algorithms of TEAR intact, but we use round-trip delays instead of packet losses to invoke these congestion control mechanisms. Our belief is that, on wireless networks, round-trip delay is a less ambiguous indication of congestion when compared to packet loss. We implemented this modified version of TEAR, evaluated its performance on IS-95B and IS-95C CDMA networks in Seoul, South Korea and we present our preliminary results in the remainder of this thesis.

It needs to be mentioned here that the delay-based modification to TEAR is an approach, which, we feel, is a promising avenue that can be further explored. The results that are presented here represent our initial efforts to use TEAR on wireless networks . Further work and design could result in more improvements in the performance of the protocol.

6.2 Congestion Avoidance Using Round-trip Times

Figure 6.1 shows the general patterns of throughput and round-trip delays as a function of the network load [8]. At low loads, the throughput increases with the load and

the round-trip delay tends to remain constant. When the load equals the network capacity, the throughput stops increasing. Beyond a certain point, queues start building up, packet losses start occurring and any further increase in the load will result in a sudden fall in the throughput. The network is now said to be congested. The round-trip delay also follows a similar pattern. At low loads, an increase in load results only in a marginal increase in the delay. When the queues start building up, the delay increases linearly with load and after *knee* point, the increase is drastic.

The objective of TEAR over wireless networks is to adapt its sending rate in response to the observed round-trip delay variations and make the network operate at or to the left of the knee point, without driving it beyond the knee point into the zone where packet losses and high delays are the rules, rather than the exceptions.

6.3 Constant Bit-Rate Transmission

The first question we asked ourselves was whether there really is a need for congestion control on wireless networks. There is a certain amount of resource provisioning done by wireless providers to ensure that a basic quality of service is provided most of the time to the end user. It is also expected that, unlike the Internet backbone, the number of users who will share these resources will be limited. In spite of these facts, there is a strong motivation to do congestion control, as our following results show.

The alternative to not doing any kind of congestion control is sending at a constant predetermined rate, i.e., blasting off. If an application *blasts-off*, it needs to decide at what rate to transmit. In deciding this, the application has to take into account the following considerations:

- What is the maximum rate that can be supported by the network? If the application overshoots this rate, it is going to be penalized with heavy losses and high response times. Also, the network will be loaded beyond the knee point.

- What is the minimum rate that the application requires in order to deliver acceptable user quality? The sending rate should not fall below this, for obvious reasons.

- How is it going to interact with other flows that might be sharing network resources? Even if the number of such flows is limited, will the application's sending rate impact

on other flows? What will be the influence of other flows on the application's own flow?

To find an answer to these questions, we tested the performance - in terms of packet loss and end-to-end delay - of constant rate UDP flows over an IS 95C CDMA network provided by the South Korean wireless provider Korean Telecom-Freetel (KTF). The network provides a data service at 144kbps, but our experiences show that the effective throughput that can be achieved is much lower.

Figure 6.3 and figure 6.4 show the observed round-trip times for the packets with a sending rate of 30kbps and 60kbps respectively. We observe that the conditions are good, the loss rate is low and the RTT is between 500-800 ms most of the time. Such high RTTs are common in wireless networks and arise primarily due to the high latency of wireless links. Similar loss and delay patterns were also observed for lower sending rates.

As the sending rate is increased, we observed that the network was driven into overdrive and this resulted in a dramatic increase in the loss rates and end-to-end delays. Figure 6.5 shows the results for a run with a constant sending rate of 80kbps. We find that the network goes into persistent congestion quite often and the RTTs are above 1s most of the time. Packet losses increase and there are longer and more frequent burst losses too.

An application developer who sees the previous two graphs might think it is all right for him to send at a constant rate of 60kbps, since it is not going to drive the network into the danger zone. But his optimism is unfortunately misplaced because the performance he can get out of his flow will depend on the presence of other flows in the network, however small the number of such flows is going to be. We ran four 30kbps flows simultaneously, making them share the network resources. Figure 6.6 shows the loss-delay characteristics of one of the flows. Each individual flow is unlikely to have overloaded the network, but their combination has and this results in a degradation of performance for each of the flows. We see that the RTTs have shot up well beyond 1 second when compared to a single 30kbps run and the loss rates have also increased.

We conclude that the alternative to not performing congestion control - sending at constant rates - is not flexible, is unstable and can lead to oversubscription and under-utilization of the network.

6.4 Packet Loss and RTT As Congestion Indicators

The second question we asked ourselves was about the scenarios in which using end-to-end delay as an indicator of congestion would be advantageous as compared to using packet losses for congestion control.

Figure 6.7 and Figure 6.8 are results from constant rate UDP runs of 30kbps and 60kbps respectively. From the round-trip times and our knowledge that the network can support these bitrates, we can safely conclude that the network does not reach a congested state during these runs. In spite of this, we see lengthy burst losses during the transmissions. We can consider these losses to be wireless link losses caused by fading, interference or path loss. Such losses are likely to happen due to busy hour traffic, adverse weather conditions, obstacles in the signal path and various other environmental and mobility factors.

If we were to react to such packet losses by halving the sending rate (or congestion window), the effect would be a drastic reduction in performance. For instance, in TCP, such burst losses would most likely cause TCP to timeout and slow-start with a congestion window of one, a severe penalty. It is also probable that TCP finds it impossible to proceed under such adverse conditions.

The round-trip times, on the other hand, do not increase dramatically before the burst loss, in fact they fall within the range that is characteristic of an uncongested network. Immediately after the burst loss, the round-trip times continue to have similar values. This lends more credence to our assertion that these losses were wireless losses and not caused by congestion. If they were caused by congestion, we must have observed a drastic increase in the round-trip times, which is not the case. A scheme which uses round-trip delay as the congestion indicator would not react to such signal fades and its performance would not be degraded by these losses.

It is also possible to distinguish between wireless losses and congestion losses by playing a waiting game. If the sender waits to find out the length of a burst loss and tries to make an intelligent decision based on this, then it is possible to filter out some of the noise caused by lossy links. On the Internet, bursts typically do not span more than a few RTTs. The sender could assume that burst losses whose length is beyond a certain threshold are wireless losses and desist from responding to them. But the sender would then have to wait even for non-bursty congestion losses to ascertain that they are not wireless losses, and this would increase its response time to react to congestion, definitely not a desirable property.

It also needs to be mentioned that using round-trip time as a congestion indicator is not a fool-proof mechanism to discriminate between link and queuing losses. Many wireless networks perform link-level retransmissions to recover from losses and this might cause an increase in the packet latency. In such cases the sender would mistake the increase in RTT to be indicative of congestion and take corrective action when none is required. We have not observed such cases frequently in our experiments and they do not seem to warrant protocol modifications.

6.5 TEAR's Delay Hysteresis Loop

For our experiments over wireless networks, we have kept the window and rate smoothing algorithms of TEAR intact, but we make the modification that instead of using packet loss, we use the round-trip time as an indication of network congestion. The fundamental idea behind TEAR remains the same - keep smoothly probing for available bandwidth until the observed latencies indicate that the network is being driven into a congested state, in which case cut down the rate without incurring the drastic fluctuations of TCP.

Traditional delay-based congestion control algorithms [8] use the delay-window gradient to make control decisions. Unfortunately these schemes work well only in deterministic networks which can be modeled with queuing servers having constant service times. There has not been much work to extend these algorithms for real networks which are probabilistic in nature.

We use a simpler approach which does not track the delay-window gradient. Instead, TEAR tries to keep the network within the safe region, i.e., to the left of the knee point. When it finds that the round-trip delays are increasing beyond a certain limit, it invokes the window decrease algorithms. The sender responds by cutting down its sending rate and this should result in a lowering of the RTTs.

At first we experimented with a single RTT threshold. Whenever the RTT falls below beyond this threshold, the window grows, and when the RTT increases beyond the threshold, the window shrinks. This rudimentary scheme is sufficient to keep the network in the safe region, but it results in rate oscillations because the network makes rapid back and forth transitions between the regions that lie on either side of the threshold.

To overcome the oscillatory behavior, we introduce the idea of a delay hysteresis

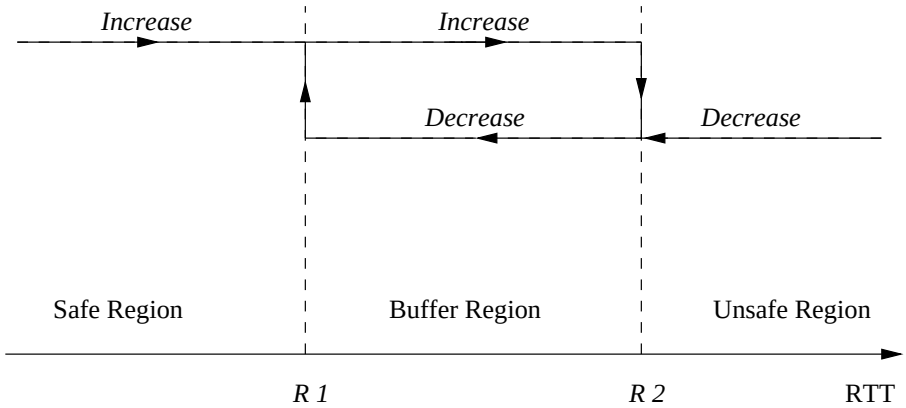

Figure 6.2: The TEAR Delay Hysteresis Loop

loop. Merriam-Webster's Collegiate Dictionary defines hysteresis thus:

hys-ter-e-sis a retardation of an effect when the forces acting upon a body are changed (as if from viscosity or internal friction); *especially* : a lagging in the values of resulting magnetization in a magnetic material (as iron) due to a changing magnetizing force.

Hysteresis is a common phenomenon in physical systems and it represents the history dependence of physical systems. The term is most commonly applied to magnetic systems and is the operating principle behind magnetic tape storage - because the magnetization lags behind the field from the tape head, when the field drops to zero, the tape stays magnetized, thereby storing data.

Hysteresis loops happen when a system is repeatedly wiggled back and forth. This is exactly what we are trying to do with the network - make it move back and forth between two regions based on the observed RTTs. 6.2 shows the delay hysteresis loop we use to avoid rate oscillations in TEAR.

When the RTT is in the region to the left of *R1*, TEAR's response is to increase its window and when the RTT is in the region to the right of *R2*, TEAR responds by reducing its window. Finally, when the RTT is in the hysteresis region between *R1* and *R2*, TEAR's response depends on the *history* of the network. If the current RTT was reached in the process of a window increase, TEAR increases the window further. If the current RTT was reached in the process of a window decrease, TEAR decreases the window further. The response signal (window increase) lags behind the control signal (the round-trip time) when approaching from the direction of *R2* and hence we say that the loop exhibits hysteresis.

Epoch	k	k - 1	k - 2	k - 3	k - 4	k - 5	k - 6	k - 7
Weight	1/2	1/4	1/8	1/16	2/32	1/64	1/128	1/128

Table 6.1: Optimized TEAR weights used for smoothing the esimated rate

The idea is to introduce a buffer between the safe and unsafe regions in order to absorb the noise signals that might occur during transitions between the two. This buffer region helps in avoiding window size oscillations.

For our implementation we have used fixed values for *R1* and *R2*, with *R1* = 800ms and *R2* = 1200 ms. This makes the protocol simple and predictable, but the scheme does not offer enough flexibility to seamlessly migrate between various networks which have different delay characteristics. The packet latency on wireless networks depends on a whole range of factors ranging from link-level retransmission mechanisms to transport-level data interleaving. Thus it can be expected to vary widely between different networks. Modifying the TEAR scheme to adapt to varying network conditions and heterogeneous networks and have dynamically varying hysteresis thresholds is work for the future.

6.6 TEAR Weights

It is important to consider performance penalties when running receiver based flow control algorithms for wireless end-devices. Many of these devices like cellular phones and person digital assistants run on low-power processor and would benefit from a light protocol overhead. Toward this end, we modified the weights that TEAR uses to smoothen its rate samples. We use weights that are powers of two, so that performance can be improved using shift operations instead of expensive division operations. 6.1 is the modified weight table.

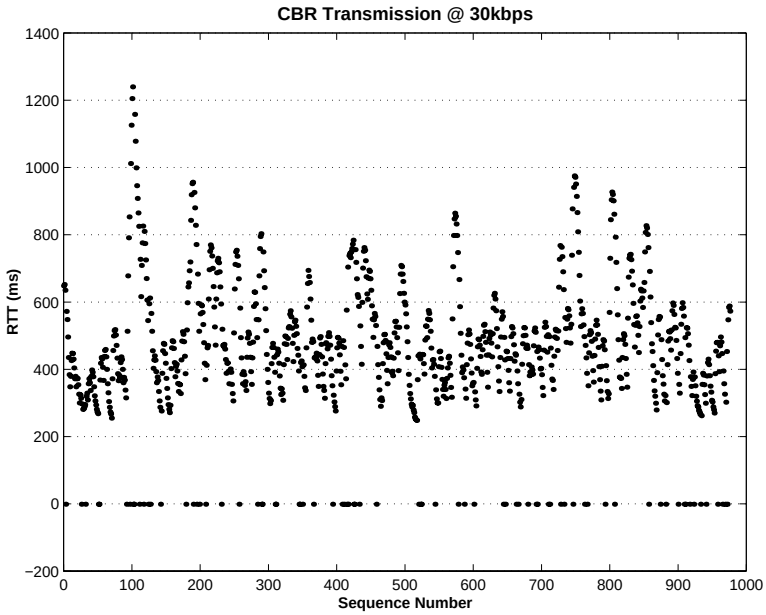

Figure 6.3: Packet round-trip times with a constant sending rate of 30kbps, IS-95C network

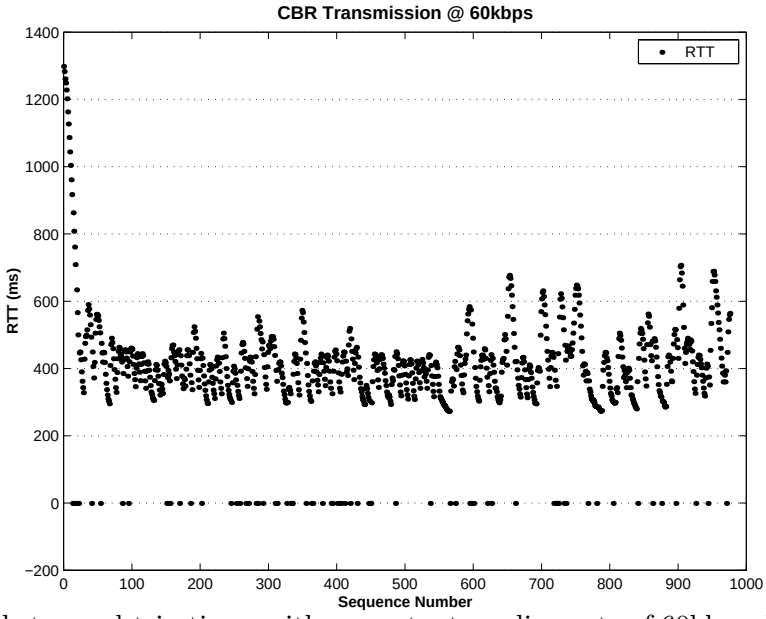

Figure 6.4: Packet round-trip times with a constant sending rate of 60kbps, IS-95C network

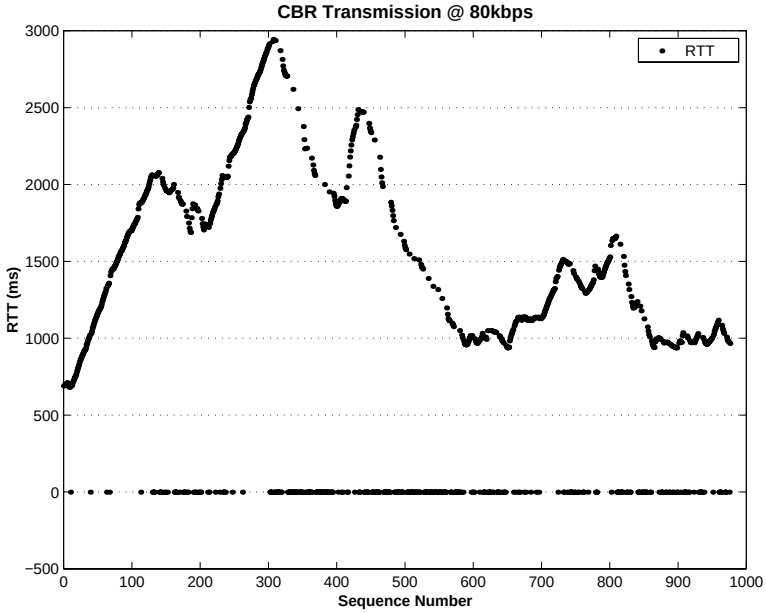

Figure 6.5: Packet round-trip times with a constant sending rate of 80kbps, IS-95C network

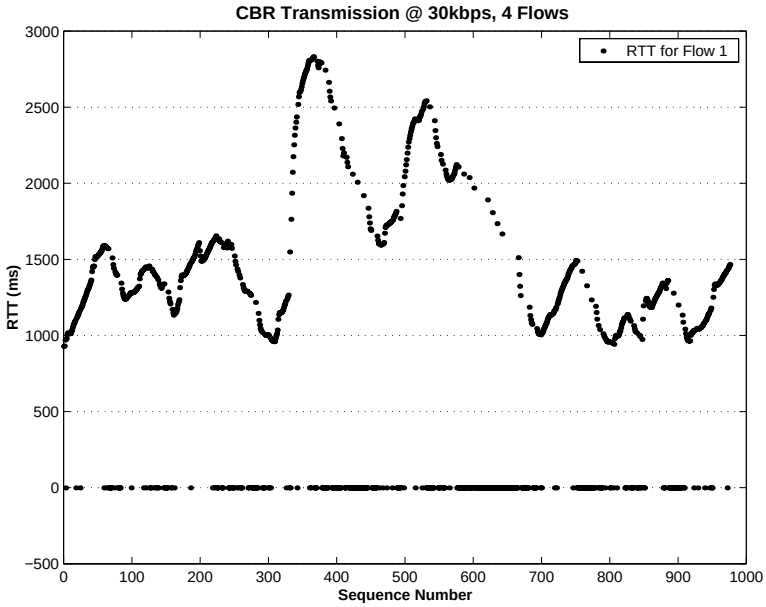

Figure 6.6: Packet round-trip times with a constant sending rate of 30kbps with 4 UDP flow sharing the IS-95C network

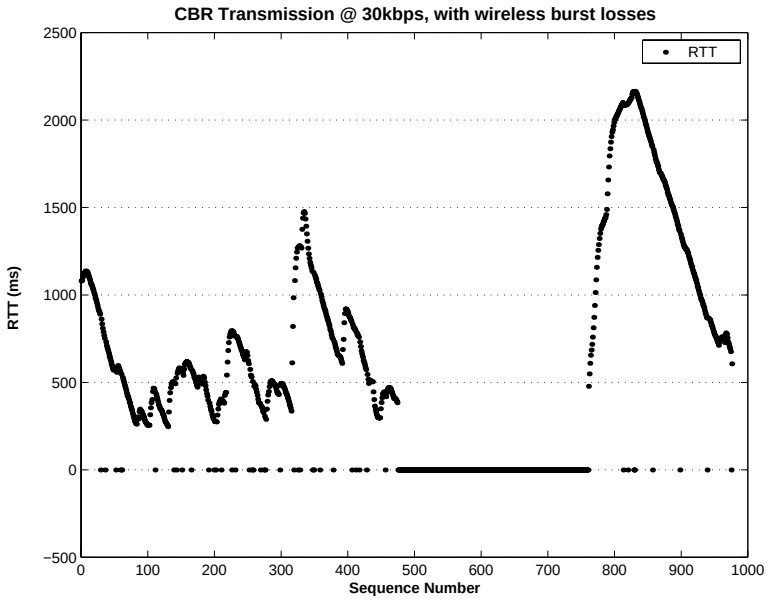

Figure 6.7: Packet round-trip times with a constant sending rate of 30kbps, illustrating the presence of wireless burst losses

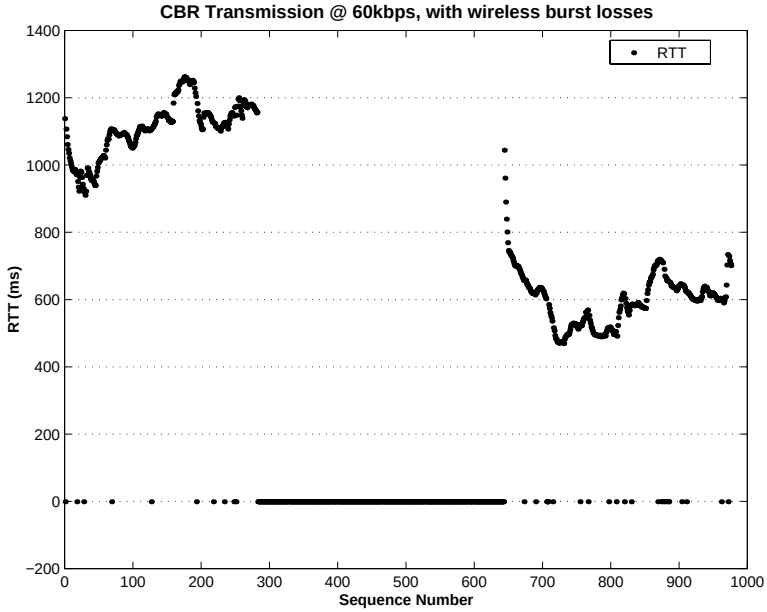

Figure 6.8: Packet round-trip times with a constant sending rate of 60kbps, illustrating the presence of wireless burst losses

Chapter 7

TEAR Over Wireless Networks: Experimental Results

7.1 Experimental Setup

The main objectives of our experiments for evaluating and verifying the performance of TEAR on wireless networks are the following:

- To observe the performance of delay-based response to congestion on wireless networks.
- To ascertain that TEAR is *fair* and *TCP-friendly*, i.e., study how TEAR flows share bandwidth with each other and how they share bandwidth with TCP.
- To study the *rate smoothness* of TEAR and compare it to the smoothness it achieve on the Internet.
- To study the throughput improvements over TCP that is obtained by using delay-based response to congestion in place of packet losses.

We conducted our experiments over two different wireless networks. One, an IS-95B wireless network provided by the Korean wireless carrier SK Telecom, with a 64kbps data service. Two, an IS-95C network provided by the carrier Korean Telecom-Freetel, with a 144kbps data service. Both TEAR and TCP were run on Microsoft Windows and the clients were connected to the wireless network through a cellular phone used as a modem.

The experiments were run through the week of June 14 - June 21, 2001 during various times of the day. In this chapter, we take a look at some of the salient and representative results that were obtained during these runs.

In evaluating and verifying TEAR's performance over wireless, we look at the following scenarios.

- Single TCP run, to find out how TCP performs over wireless. Unlike the Internet the bandwidth TCP achieves might not be the actual available bandwidth because of network underutilization.
- Single TEAR run - to find out how TEAR performs over wireless and to study how it adapts to congestion based on round-trip delays.
- Multiple TEAR flows sharing the network - to observe how TEAR shares compete with each other.
- Multiple TCP and multiple TEAR flows sharing the network - to observe how TCP and TEAR share the available bandwidth and study the TCP friendliness of TEAR.
- Bulk transfer using TEAR and TCP - to observe how much of a performance improvement TEAR provides over TCP.

We performed the comparisons between TEAR and TCP on a 64kbps data network. The bulk transfer experiments were performed on a 144kbps network.

7.2 Finding the Available Bandwidth

Single TCP Flow Figure 7.2 shows the instantaneous rate variations and aggregate throughput of a single TCP flow running over a 64kbps network. We see that that the rates fluctuate widely, quite often and fall down to zero a few times. The average throughput reached by TCP is about 30kbps. There is no way to ascertain whether this is the actual available bandwidth, but it is highly likely that because of the problems discussed before, TCP ends up under-utilizing the network.

Single TEAR Flow Figure 7.1 shows the instantaneous rate variations and aggregate throughput of a single TEAR flow running on a 64kbps network. TEAR keeps increasing

its sending rate until the measured round-trip times exceed the threshold R2. Thereafter, it decreases its window to bring the network back into the safe region below the threshold R1 and starts probing for bandwidth again. The process keeps repeating and results in a number of triangular regions formed by successive increase and decrease runs.

Effective Bandwidth of the Network The average throughput TEAR reaches is about 40kbps, higher than that of TCP. Experiments with constant bit rate sources indicate that the maximum bandwidth that can be sustained without incurring huge delays and high packet loss rates, i.e, without driving the network into the unsafe region, is about 50kbps. This maximum achievable rate may be thought of as the effective bandwidth the 64kbps network provides. TEAR falls short of this effective throughput because it is somewhat conservative and starts decreasing its rates before the network reaches a congested state. It is possible to nudge the throughput closer to the effective bandwidth by altering the hysteresis thresholds, but this in turn increases the risk that the network could be driven into persistent congestion, especially when multiple TEAR flows are running together. We have adopted the more conservative approach of keeping the network in a stable state at the expense of higher throughput.

7.3 Comparison with TEAR over the Internet

If we compare the rate variations of TEAR over wireless with those of TEAR over the Internet, there are two observations we can make.

Predictability One is that the frequency of rate oscillations is quite small, as is evidenced from the small number of triangular regions in the graphs. The rates typically do not oscillate over small timescales, instead they tend to swoop up and down over larger time intervals spanning a few seconds. This makes the behavior of TEAR more predictable than its behavior over the Internet, where the oscillations around the fair share are more frequent. This predictability could be of great benefit to end applications, which can make provisions for the times during which data supply from the network might dry up.

An Example: Quality Adaptation for Video Playback One example would be quality adaptation for video playback [23]. Quality adaptation mechanisms add and drop

layers of the video stream and adapt the compression to fit into the varying available bandwidth. Such schemes perform TCP-friendly congestion control to react to congestion over short timescales and use layering to perform coarse-grain adaptation. The objective is to provide smoothly varying quality to the end user. In order to execute an effective add-and-drop algorithm, the sender needs to estimate the amount of buffering that the receiver has at any point in time. This estimation is normally based on the assumption of an additive-increase-multiplicative-decrease TCP-friendly congestion control algorithm. But in real networks, since the rate fluctuates over smaller timescales, a long term estimation of the buffering at the receiver becomes difficult. This makes a conservative add-and-drop policy for layers, which assumes that conditions will go bad in the near future, necessary. The result is a lowering of the end quality that is delivered. But with the kind of rate variations that TEAR exhibits over wireless, however, it becomes possible to do such longer-term estimates about end-to-end data delivery with greater accuracy. Applications can afford to be less conservative in their assumptions of the behavior of the network in the immediate future and this could help in improving the quality delivered to the end user.

Dynamic Range The second observation that can be made in comparison with TEAR over Internet is that the dynamic range spanned by the TEAR rates is higher. In Figure 7.1, the rates vary from around 25kbps to around 75 kbps, a wider range compared to the smooth variations around the fair-share we can observer on the Internet. The reason for this wider devitaion is the asymmetry and hysteresis involved in TEAR's response to the congestion. Once TEAR decides that the network is congested and has entered the unsafe region, it keeps decreasing its rates until the network comes back into the safe region. This process takes some time because the network has to go through the buffer region before being deemed as safe. Each decision by TEAR to cut its rate takes one round-trip time to have its effect on the network. Since round-trip times are more than a second when the network is congested, it generally takes a few seconds for the network to settle back into the safe region. Throughout this interval, TEAR decides that the network is operating in the buffer region, causing a sequence of rate cuts. The higher number of successive window decreases results in a higher fall of the estimated rate from its peak point than compared to the Internet. Although the fall is higher in terms of magnitude, it is not necessarily steeper, because the it normally happens over a longer interval. Thus TEAR still exhibits a certain smoothness in its rate variations. In Wall Street parlance, we can say that TEAR effects a

soft-landing when it detects network congestion.

7.4 Multiple TEAR and TCP Flows

First we try to find out how TEAR flows compete with each other. Figure 7.3 and 7.4 show the instantaneous rate variations and aggregate throughputs respectively of 2 TEAR flows running together and sharing the 64kbps network. The rate variations of both the flows are similar and consist of a sequence of triangular regions. The aggregate throughput obtained by the flows is just over 20kbps, which can be considered to be the fair-share because a single TEAR flow running alone achieves a maximum throughput of 40kbps.

Figure 7.5 and Figure 7.6 show the instantaneous rate variations and aggregate throughput respectively of one TEAR flow and one TCP flow running together. The variations in TCPs rate are drastic, sometimes the rate shoots up to 100kbps and many times it falls to zero. Most of the time, however, the rate falls below 20kbps and the throughput achieved by TCP is about 15kbps. This is much lower than the 25kbps that TEAR is able to get from the network. The rate variations of TEAR are smoother too, when compared to TCP. Figure 7.7 and Figure 7.8 are similar graphs for 2 TEAR and 2 TCP flows sharing the network. Again, we observe that TCP's throughput is almost half as that of TEAR, while TEAR flows get a throughput of around 15kbps, TCP manages only 8kbps.

TCP Friendliness The concept of TCP-friendliness on wireless networks is somewhat ambiguous because even in the absence of competing flows, TCP often ends up under-utilizing the network. When there are competing flows, it becomes difficult to filter out the reasons for TCP's underperformance - it could be due to link errors or due to starvation by more aggressive competing flows. Hence the definition of TCP-friendliness cannot be transplanted directly from the Internet scenario to the wireless case. Instead, what we measure here is whether or not TCP performance is "reasonable" when it competes with TEAR, i.e., whether or not TCP is driven to the ground by TEAR. Our experiments show that in this loose sense, TEAR is friendly to TCP, although the TCP throughput often falls below the fair-share of the effective bandwidth. It is hard to sift out the effect TEAR has on TCP from the reasons inherent in TCP that cause it to underperform. The definition of TCP friendliness over wireless networks and whether it has the same importance as it has

on the Internet is an open question.

7.5 Reliable Bulk Transfer

Since achieving a higher throughput than TCP is an equally, if not more, important consideration over wireless networks than TCP-friendliness or rate smoothness, we performed experiments to compare the throughputs of TEAR and TCP for a reliable bulk data transfer application.

TEAR in its barebones form provides only a rate-control module for an application. In TCP there is a close coupling between the rate control and reliability parts of the protocol. But this is not the case with an application that uses TEAR. If reliability is desired, it is necessary to build a module which will detect packet losses, do retransmissions and timeouts, control the receiver's buffer and perform other functionalities that are essential for ensuring end-to-end reliability.

We built a simple reliable protocol over TEAR, which relies on selective acknowledgments and timeouts for recovering lost packets. We use this protocol to find out how reliable transfer over TEAR compares with reliable transfer over TCP. The experiments were run over an IS-95C network which provides a data service of 144kbps.

Table 7.1 and Table 7.2 list the transfer times and throughputs for 20KB, 50KB and 100KB bulk transfers using TCP and TEAR respectively. Figure 7.9, 7.10 and 7.11 graphically illustrate the distribution of transfer times for each of these transfers. The results are from experiments run on June 20, 2001 at various times of the day.

We are interested in the throughputs achieved by the two protocols and the consistency with which they achieve their respective throughputs. Constant bit-rate experiments indicate the maximum throughput that can be achieved without incurring inordinate delays and high packet loss rates on the IS-95C network is about 70kbps. This is the maximum effective bandwidth of the network, but under busy hours and adverse environmental conditions, this can be much lower.

The throughput difference between TEAR and TCP increases with an increase in the bulk size. At 20KB, TCP's throughput is comparable with that of TEAR. When the network conditions are good, TCP is sometimes superior because of its more aggressive bandwidth probing. But even for short transfers TCP cannot be trusted to provide this performance consistently. There are cases where there is a severe degradation in TCP's

Time of day	20 KB		50 KB		100 KB	
	Time (s)	Rate (bps)	Time (s)	Rate (bps)	Time (s)	Rate (bps)
10:00-11:00	3	47962	27	14637	32	25037
	3	50630	38	10537	68	12011
	9	17291	21	19174	35	23022
	11	14831	20	20317	70	11539
	4	38040	12	33115	28	29162
13:00-14:00	2	56399	14	27560	19	42033
	5	27397	5	70029	37	21830
	6	24786	17	23640	34	23883
	5	28640	6	65958	31	25755
	98	1671	22	18523	45	18137
16:00-17:00	26	6102	56	7247	172	4748
	62	2604	38	10527	250	3275
	24	6760	33	12296	102	7974
	161	1013	25	8776	193	4224
	3	49408	14	27854	238	3430
19:00-20:00	3	45184	76	5371	35	22768
	44	3685	67	6102	90	9031
	31	5267	54	7506	64	12727
	41	3955	25	15779	43	18904
	3	43218	113	3619	109	7447

Table 7.1: TCP bulk transfer throughput for 20KB, 50KB and 100KB transfers

throughput - these can be seen as spikes in Figure 7.9. For 50KB and 100KB transfers, the performance difference between TEAR and TCP becomes wider. As a rule-of-thumb, if we define that whenever the throughput is about 15kbps, the protocol is suffering from a severe degradation in performance, then TCP experiences this condition more often as the transfer size increases. TEAR is not immune from such performance degradations either, but it is more consistent and predictable than TCP in achieving higher throughputs under varying network conditions and during different times of the day.

Time of day	20 KB		50 KB		100 KB	
	Time (s)	Rate (bps)	Time (s)	Rate (bps)	Time (s)	Rate (bps)
10:00-11:00	4	40960	18	22755	51	16062
	15	10922	16	25600	18	45511
	7	23405	7	58514	25	32768
	3	54613	24	17066	16	51200
	3	54613	31	13212	28	29257
13:00-14:00	4	40960	27	15170	16	51200
	5	32768	10	40960	17	48188
	3	54613	6	68266	23	35617
	28	5851	11	37236	24	34133
	12	13653	13	31507	25	32768
16:00-17:00	9	18204	18	22755	53	15456
	4	40960	17	24094	25	32768
	5	32768	58	7062	40	20480
	20	8192	23	17808	32	25600
	12	13653	15	27306	18	45511
19:00-20:00	4	40960	25	16384	35	23405
	5	32768	19	21557	50	16384
	6	27306	32	12800	19	43115
	7	23405	15	27306	26	31507
	3	54613	26	15753	78	10502

Table 7.2: TEAR bulk transfer throughput for 20KB, 50KB and 100KB transfers

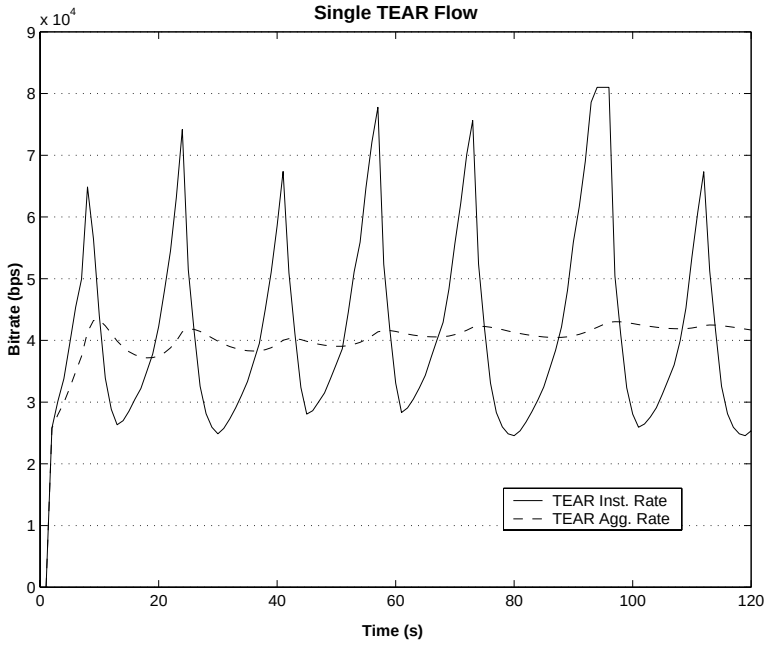

Figure 7.1: Instantaneous rate and aggregate throughput of a single TEAR flow

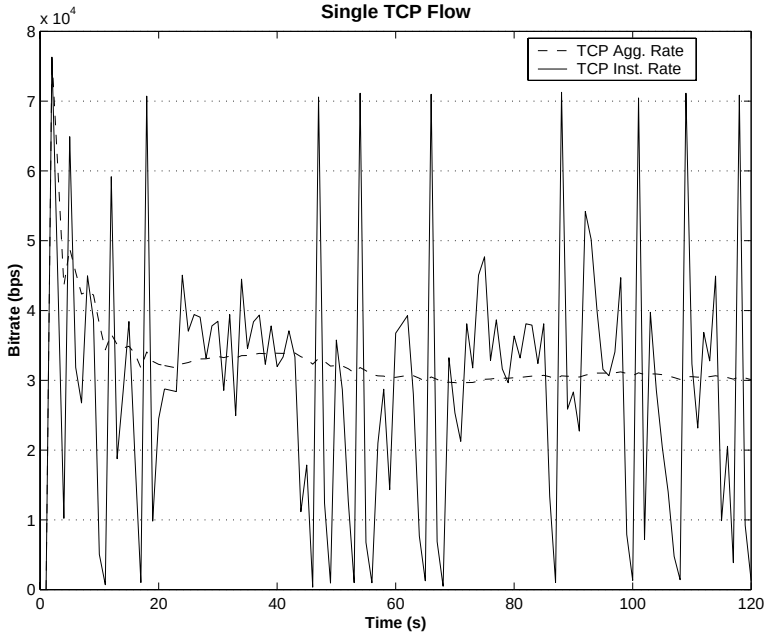

Figure 7.2: Instantaneous rate and aggregate throughput of a single TCP flow

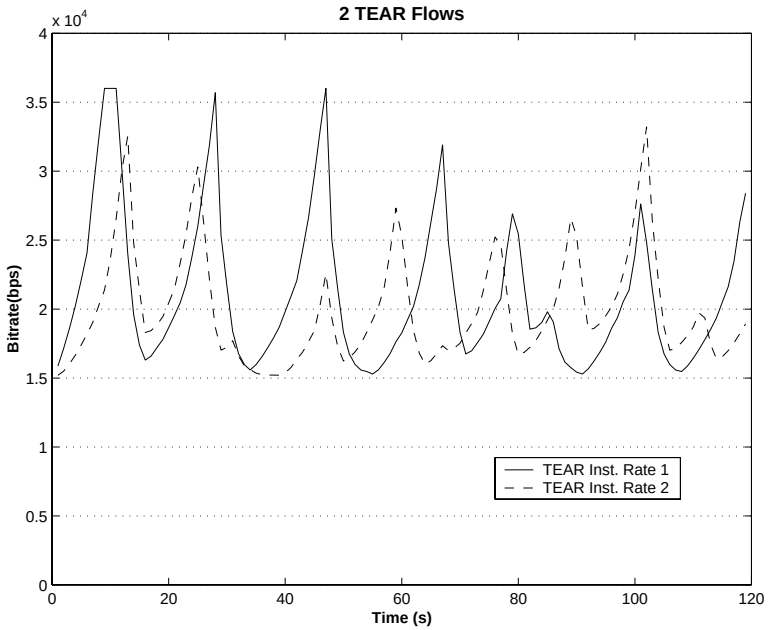

Figure 7.3: Instantaneous rates of 2 TEAR flows sharing the network

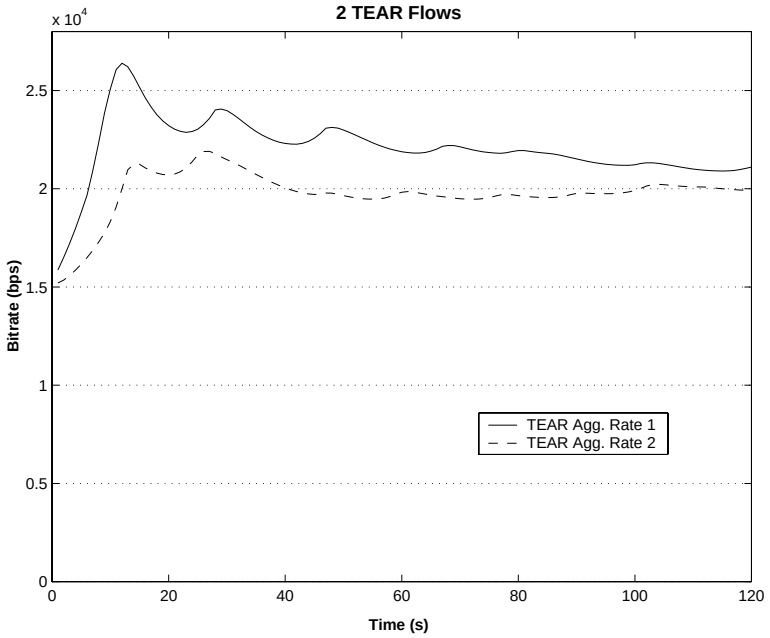

Figure 7.4: Aggregate throughputs of 2 TEAR flows sharing the network

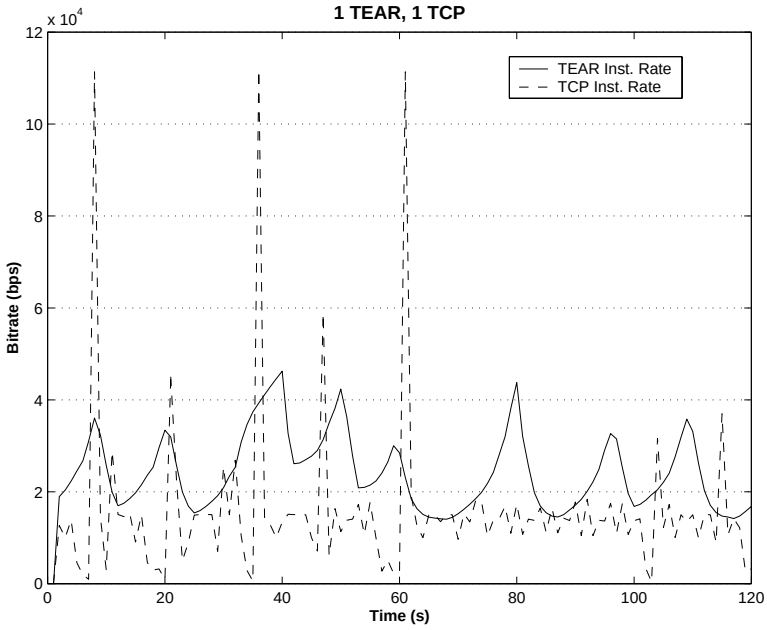

Figure 7.5: Instantaneous rates of 1 TEAR flow and 1 TCP flow sharing the network

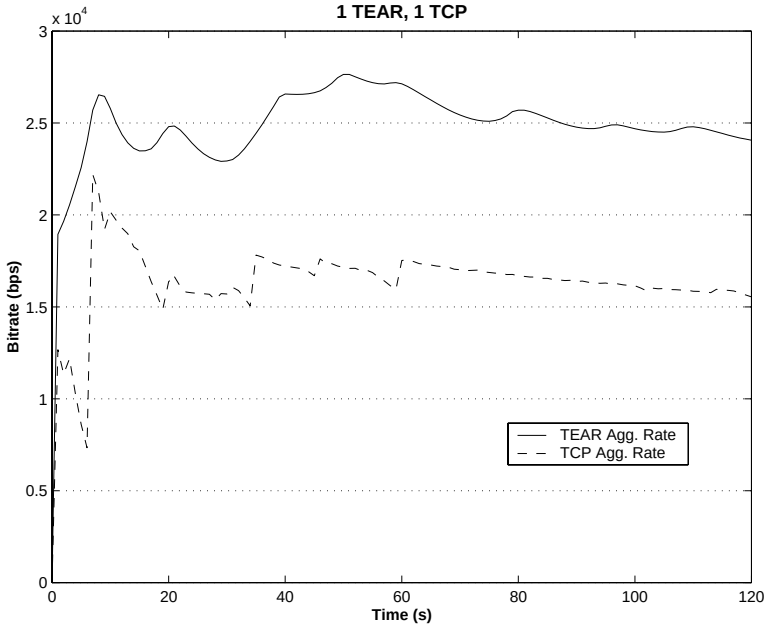

Figure 7.6: Aggregate throughputs of 1 TEAR flow and 1 TCP flow sharing the network

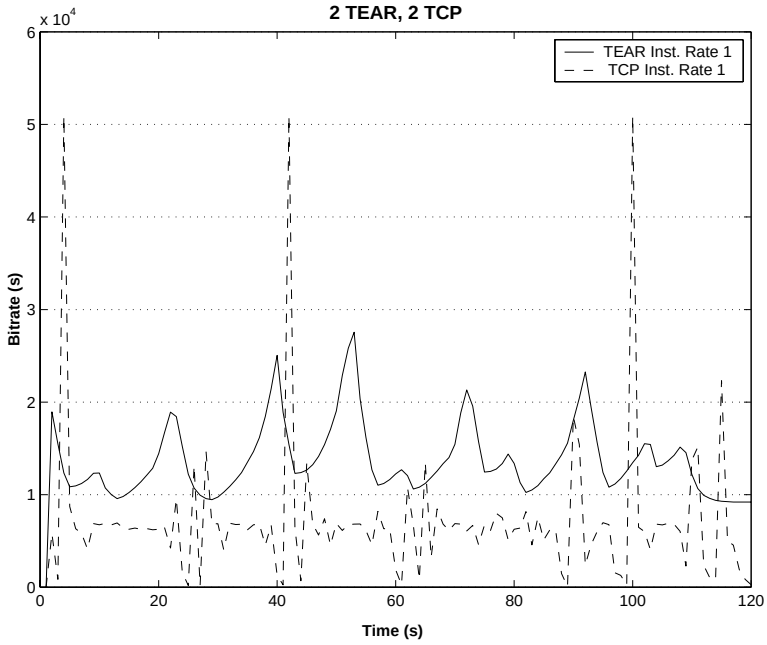

Figure 7.7: Instantaneous rates of representative TEAR and TCP flows

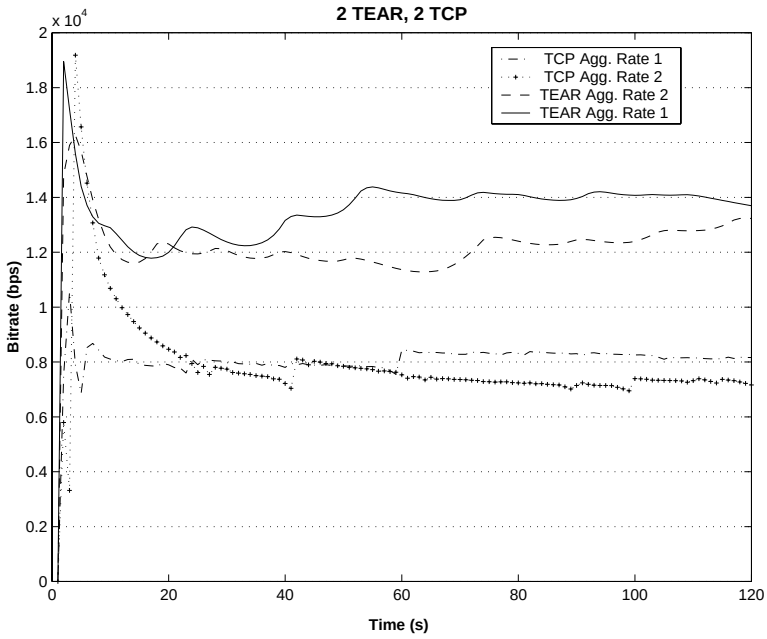

Figure 7.8: Aggregate throughputs of 2 TEAR and 2 TCP flows sharing the network

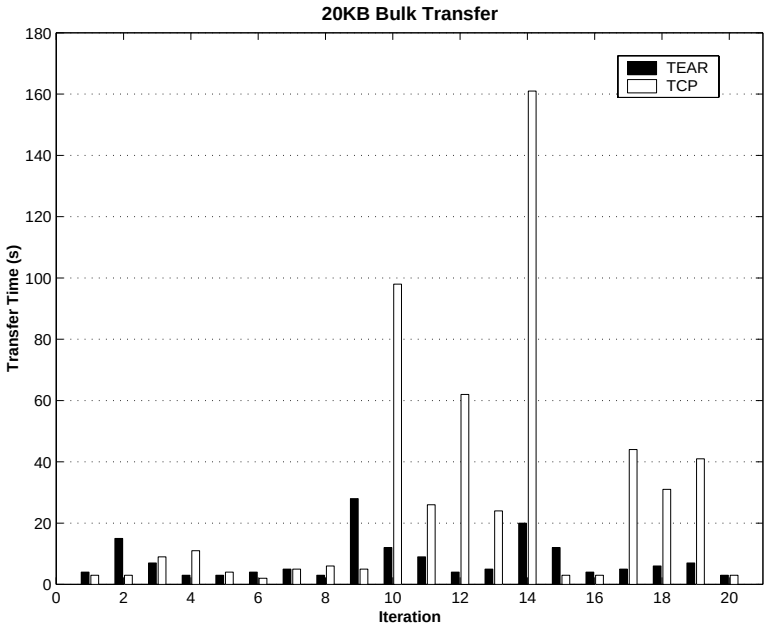

Figure 7.9: Comparison of TEAR and TCP transfer times for 20KB reliable bulk transfer

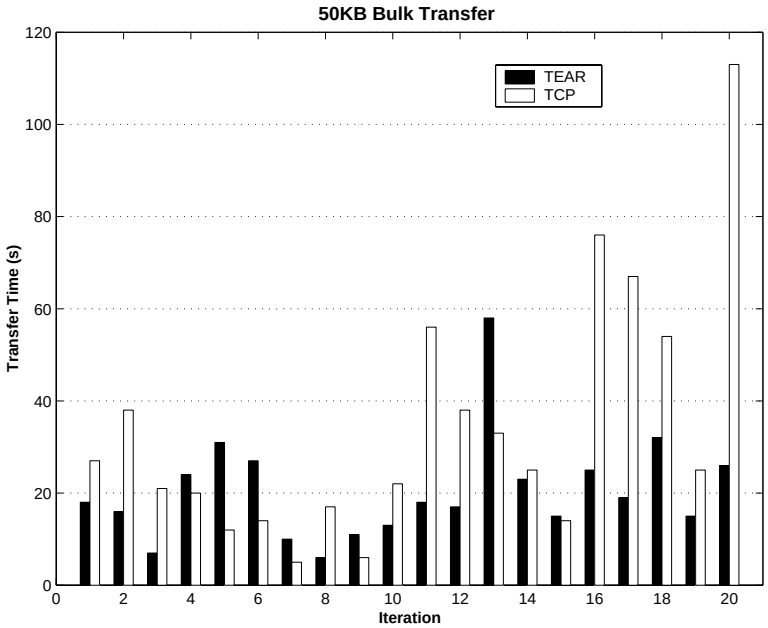

Figure 7.10: Comparison of TEAR and TCP transfer times for 50KB reliable bulk transfer

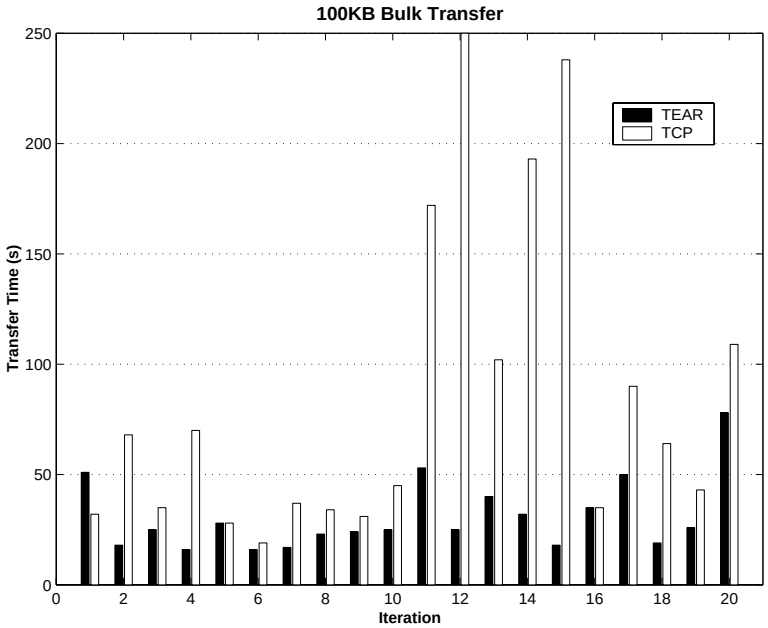

Figure 7.11: Comparison of TEAR and TCP transfer times for 100KB reliable bulk transfer

Chapter 8

Conclusion

In this thesis, we have presented and analyzed the results from experiments of running TEAR, a TCP-friendly flow control protocol for multimedia streaming, over the Internet and wireless networks.

Modern day, bandwidth hungry applications like multimedia streaming find TCP's response to congestion too severe. There is a looming threat of congestion collapse from such applications that do not do any kind of congestion control. The requirement is for protocols that are application-friendly, by providing smooth variations in their sending rates, and network-friendly, by responding to persistent network congestion. TEAR is one such protocol. We first described the TEAR protocol [24] in detail touching upon the various design aspects. Then we presented the results of our experiments with TEAR over the wired Internet. We showed that TEAR is TCP-friendly, shows superior rate smoothness to TCP and that TEAR flows compete fairly with each other and with TCP. In order to be a protocol viable to be deployed on real networks, the stability of a protocol is important. We showed that TEAR is stable, in the sense that, in the presence of other flows joining and leaving, TEAR eventually reaches its fair share of bandwidth when the network settles down to a steady state.

Wireless networks have different characteristics from the Internet and provide different challenges. One of the main problems protocols face on wireless networks is the inability to distinguish packet losses caused by lossy wireless links from those caused by network congestion. This severely degrades the performance of protocols like TCP. We described how TEAR can be used over wireless networks, by modifying it to use round-trip delays instead of packet losses as an indication of congestion. We presented and analyzed

our preliminary results from running TEAR over commercially deployed wireless networks in South Korea. We showed that TEAR achieves superior throughput compared to TCP. It also exhibits lower fluctuations in its sending rate and its behavior is more predictable than its behavior on the Internet. TEAR is friendly to TCP over wireless networks, in the sense that it does not drive TCP to starvation. Finally we presented our results comparing the throughputs for bulk transfer achieved by TCP and a reliable protocol we implemented over TEAR.

A lot of work can be done to improve the performance of TEAR over wireless networks. Modifying the simple hysteresis based approach of reacting to congestion to a more generic delay-gradient based approach is worth considering. The protocol also needs to be tested and made adaptable to work over various wireless environments which might have vastly differing delay characteristics.

In this thesis, we have not investigated the integration of quality adaptation schemes with TEAR. We believe it is possible to incorporate such schemes to provide a smoothly varying end-user quality. This avenue of coupling TEAR flow control with differing application specific requirements needs to be explored further.

Bibliography

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

[2] H. Balakrishnan, R. Katz, V. Padmanabhan, and S. Seshan. "Improving Performance of TCP over Wireless Networks." Technical Report, Texas A&M University, 1996.

[3] R. Braden. "Requirements for Internet Hosts - Communication Layers". IETF RFC 1122, October 1989.

[4] H. Balakrishnan, S. Seshan, and R. H. Katz. "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks". ACM Wireless Networks, 1(4), December 1995.

[5] S. Bhattacharya, D. Towsley, and J. Kurose. "The Loss Path Multiplicity Problem for Multicast Congestion Control". In *Proceedings of IEEE INFOCOM*, 1999.

[6] S. Floyd, M. Handley, J. Padhye, and J. Widmer. "Equation-Based Congestion Control for Unicast Applications". ACM SIGCOMM 2000, Stockholm, Aug 2000.

[7] S. J. Golestani and K. K. Sabnani. "Fundamental Observations on Multicast Congestion Control in the Internet". In *Proceedings of the IEEE INFOCOM*, New York, NY, March 1999.

[8] R. Jain. "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks". Computer Communication Review, V.19 N.5, October 1989, pp. 56-71.

[9] S. Jacobs and A. Eleftheriadis. "Providing Video Services over Networks without

Quality of Service Guarantees". In *World Wide Web Consortium Workshop on Real-Time Multimedia and the Web*, 1996.

[10] R. Jain, K. Ramakrishnan, and D. Chiu. "Congestion Avoidance in Computer Networks with a Connectionless Network Layer". Technical Report. DEC-TR-506, Digital Equipment Corporation, August 1987.

[11] X. Li, S. Paul, and M. Ammar. "Multi-Session Rate Control for Layered Video Multicast". In *Proceedings of Symposium on Multimedia Computing and Networking*, San Jose, CA, January 1999.

[12] R. Ludwig. "A Case for Flow-Adaptive Wireless Links". Technical Report CSD-99-1053, University of California, Berkeley, 1999.

[13] R. Ludwig. "Eliminating Inefficient Cross-Layer Interactions in Wireless Networking". PhD Thesis, Aachen University of Technology, April 2000.

[14] J. Mahdavi and S. Floyd. "TCP-friendly Unicast Rate-based Flow Control". Note sent to end2end-interest mailing list, 1997.

[15] S. McCanne, V. Jacobson, and M. Vetterli. "Receiver-driven Layered Multicast". In *Proceedings of SIGCOMM*, Stanford, CA, 1996.

[16] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. "The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm". ACM Computer Communication Review, 27(3), July 1997.

[17] J. Padhye, V. Firoiu, D.Towsley, and J. Kurose. "Modeling TCP Throughput: A Simple Model and its Empirical Validation". SIGCOMM Symposium on Communications Architectures and Protocols, Aug, 1998.

[18] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. "A Model Based TCP-Friendly Rate Control Protocol". In *Proceedings of NOSSDAV'99*, 1999.

[19] J. Postel. "Transmission Control Protocol". IETF RFC 793, 1981.

[20] T. Rappaport. "Wirless Communications: Principles and Practices". Prentice Hall, 1996.

[21] I. Rhee, N. Balaguru, and G. Rouskas. "MTCP: Scalable TCP-like Congestion Control for Reliable Multicast". In *Proceedings of IEEE INFOCOM*, New York, NY, March 1999.

[22] R. Rejaie, M. Handley, and D. Estrin. "An End-to-end Rate-based Congestion Control Mechanism for Real-time Streams in the Internet". In *Proceedings of INFOCOMM 99*, 1999.

[23] R. Rejaie, M. Handley, and D. Estrin. "Quality Adaptation for Congestion Controlled Video Playback over the Internet". SIGCOMM, pp. 189-200, 1999.

[24] I. Rhee and V. Ozdemir. "TEAR: TCP Emulation At the Receivers - Flow Control for Multimedia Streaming". Technical Report, Department of Computer Science, North Carolina State University, 1999.

[25] S. Ramesh and I. Rhee. "Issues in Model-Based Flow Control". Technical Report 99-15, Department of Computer Science, North Carolina State University, 1999.

[26] H. Schulzrinne, S. Casner, R. Fredrick, and V. Jacobson. "RTP: A Transport Protocol for Real-Time Applications". RFC 1889, January, 1996.

[27] D. Sisalem, and H. Schulzrinne. "The Loss-Delay Adjustment Algorithm: A TCP-friendly Adaptation Scheme". In *Workshop on Network and Operating System Support for Digital Audio and Video*, July 1998.

[28] W. Stallings. "Data and Computer Communications". Prentice-Hall, sixth edition, 2000.

[29] T. Turletti, S. Parisis, and J. Bolot. "Experiments With a Layered Transmission Scheme Over the Internet". Technical Report RR-3296, Inria, France, 1997.

[30] W. Tan, and A. Zakhor. "Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-friendly Transport Protocol". In *IEEE Transactions on Multimedia*, 1(2):172-186, June 1999.

[31] N. Vaidya. "Tutorial on TCP for Wireless and Mobile Hosts". 1999. Available at: http://cashew.cs.tamu.edu/faculty/vaidya/seminars/tcp-tutorial-aug99.ppt.

[32] L. Vicisano, L. Rizzo, and J. Crowcroft. "TCP-like Congestion Control for Layered Multicast Data Transfer". In *Proceedings of IEEE INFOCOM*, August, 1997.

[33] B. Whetten. "Target goals for RM Congestion Control Algorithms". IRTF Reliable Multicast Research Group Meeting in George Mason, Virginia, December 1998.

[34] L. Wu, R.Sharma, and B. Smith. "Thinstreams: An Architecture for Multicast Layered Video". In *Proceedings of the Seventh International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 97)*, May, 1997.