Achieving Loss Discrimination with Congestion Avoidance in a Rate-based Protocol

Shravan Gaonkar*, Robin Kravets†
Department of Computer Science,
University of Illinois at Urbana Champaign.
gaonkar@cs.uiuc.edu*, rhk@cs.uiuc.edu†

Luiz Magalhaes
Telecommunications Engineering Department,
Universidade Federal Fluminense - Brazil.
schara@telecom.uff.br

Report No. UIUCDCS-R-2004-2434 (Engr. No. UILU-ENG-2004-1735) May 2004

Abstract

The stability of the Internet depends on the presence of well-behaved flows. While TCP forms the majority of the traffic over the Internet, the Internet user community is also dedicated to supporting mobile devices over wireless networks. However, TCP is not well suited to support applications over lossy wireless networks. Ideally, these applications require a protocol that adapts to network conditions with the ability to discriminate congestion-based and transmission-based losses.

In this paper, we present a rate-based transport protocol that is designed to support communication for mobile nodes using infrastructure-based wireless networks. We propose an end-to-end approach to loss discrimination based on network state estimation at the receiver. Discrimination is achieved by correlating short-term history of packet inter-arrival times with the loss. We also integrate a congestion avoidance algorithm to react to pending network congestion. Simulations of our protocol prototype show that rate-based protocols can provide better correlation to network conditions than ACK-clocked protocols. We provide extensive evaluation through simulation demonstrating that our protocol is well-behaved over a wide range of scenarios.

Index Terms

System design, Simulations, Transport Protocol, Congestion Avoidance, Loss Discrimination.

I. Introduction

Advances in communication technology have increased the reliability of wired networks to the point where transmission losses are rare. As a result, losses in the Internet are generally due to congestion caused by path overuse. Such specific knowledge about network reliability has enabled TCP to be tuned to assume that all losses are caused by congestion. The introduction of technologies such as radio networks with higher loss rates breaks this assumption. Some losses can be handled by making the lossy link appear reliable. For example, wireless technologies like IEEE 802.11 [1] provide link-layer error correction to mask packet loss due to transmission errors. Such an approach assumes that the transmission losses can be recovered locally in a fashion that does not adversely affect the end-to-end transmission of data. These approaches do not, however, guarantee that no transmission losses will occur. Therefore, it is necessary to adapt transport protocols to handle transmission losses while retaining important transport protocol behavior such as congestion control and fairness to other flows. The challenge to designing such a protocol lies with the fact that a communication channel may span both wired and wireless links, introducing both congestion and transmission losses into the channel.

In general, transport protocols should react to congestion losses in a fashion that helps to alleviate congestion in the network. If congestion losses are treated as transmission losses, the sender does not decrease its offered load and more congestion builds up in the network. On the other hand, losses due to transmission do not indicate congestion. Therefore, if transmission losses are treated as congestion losses, the sender unnecessarily reduces its offered load, reducing the throughput of the stream. Either type of misclassification of losses can have detrimental effects on the communication flow and on the network. These limitations demand the design of a protocol that enables distinguishing congestion losses from transmission losses.

Recently, there have been several proposals to optimize TCP for wireless networks [2], [3], [4], [5]. Some of these solutions [2], [3] try to maintain the semantics of TCP congestion control by hiding transmission losses from the end hosts by modifying the underlying infrastructure. Techniques like Explicit Loss Notification (ELN) [6] and Explicit Congestion Notification (ECN) [7] can be integrated into an end-to-end approach to provide almost perfect knowledge of the cause of a loss. However, such an approach requires the expensive replacement of routers and other support infrastructure making deployment impractical. Therefore, a precise end-to-end loss discrimination scheme is necessary to maximize the performance of a transport protocol in dynamic wired-wireless networks without infrastructure support.

In this paper, we propose and analyze the eXtendend Rate-based Transport Protocol (XRTP). The rate-based nature of XRTP provides regularity of packet transmission that enables the tracking of congestion in the network, allowing XRTP to discriminate packet losses effectively. Additionally, XRTP reacts appropriately to varying network conditions by incorporating congestion avoidance. Both loss discrimination and congestion avoidance are achieved through estimation of available bandwidth using a combination of packet-pair [8] and measurements of jitter (i.e., the difference between inter-arrival times of packets at the receiver and inter-sending times at the sender), enabling XRTP to dynamically adapt its parameters to the current network conditions. Short term jitter enables XRTP to effectively distinguish congestion losses from transmission losses, allowing XRTP to react appropriately. XRTP also incorporates a novel congestion avoidance algorithm that adapts its level of rate increase in the presence of congestion in the network. Essentially, XRTP is more aggressive when there is no congestion allowing it to compete with other aggressive protocols such as TCP, but is less aggressive in the presence of congestion, supporting the goal of good congestion avoidance. Given the potential inaccuracies in bandwidth estimation using packet-pair, XRTP integrates non-parametric density estimation [9] to filter out irrelevant measurements. These combined mechanisms allow XRTP to achieve good throughput in wired/wireless environments while maintaining fairness to competing traffic.

The XRTP design supports both bulk data transfer and multimedia traffic. However, in this paper, we present a prototype targeted at bulk data transfer. An extension to multimedia traffic would not impact the flow control or congestion control mechanisms, but simply require different reliability support with knowledge of priorities and timeliness. In addition, the rate-based nature of XRTP naturally supports multimedia traffic.

The rest of this paper is organized as follows. Section II discusses transport protocol design in the context of current research in the area. Section III presents the design of the XRTP. Section IV presents the performance evaluation of the protocol through simulations. Finally, in Section VI, we present conclusions and directions for future research.

II. Transport Protocols for Wired/ Wireless Networks

Without explicit loss or congestion notification from the network, successful loss discrimination is dependent on implicit mechanisms available at the transport layer to classify losses. Since transport protocols implicitly monitor the transmission stream for indications of congestion, similar techniques can be used to determine if there is congestion when a loss occurs. Designing an end-to-end transport protocol for wireless networks that enables effective loss discrimination requires four separate components: flow control, bandwidth estimation, congestion avoidance and loss discrimination. This section describes various issues involved with these mechanisms, providing insight into the design decisions of XRTP.

A. Flow Control

Current transport protocols use either ACK-clocked or rate-based mechanisms to control the transmission flow of a stream. ACK-clocked protocols have dominated in wired environments due to their self-clocking nature and ease of implementation. However, rate-based protocols, such as RAP [10] and TFRC [11] have become popular due to their smoother transmission when compared to the burstiness of ACK-clocked protocols. Essentially, rate-based flow control provides a regularity in transmission that supports improved monitoring of network conditions. Additionally, packet losses due to congestion are reduced since router queues are uniformly filled by regular packet transmissions instead of overflowing from bursts. While rate-based protocols have been proposed for wired networks, current protocols are not directly applicable to wireless networks. RAP mimics the additive increase/multiplicative decrease congestion control scheme of TCP. However, this approach inhibits RAP from reacting to rapidly changing bandwidth as expected in wireless networks. TFRC is a TCP throughput equation-based rate-control protocol that includes packet loss rate and round trip time in the congestion control algorithm. The losses experienced by a flow shapes the throughput of TFRC. Without explicit loss discrimination, high loss rates degrade the performance of TFRC.

There have also been several protocols proposed to enhance TCP with packet-spacing to exploit the advantages obtained by transmitting packets at regular intervals while maintaining the semantics of TCP [12], [13]. Even though packet-spacing alleviates the bursty nature of TCP, it does not decouple the protocol's reliability mechanisms from congestion control. Retransmissions in TCP for reliability could potentially stop the transmission of packets, limiting the benefits gained by packet spacing. Wireless-TCP [14] is a reliable end-to-end rate-based transport protocol designed to operate on wireless wide-area networks. WTCP tracks the ratio of inter-packet separation at the receiver to inter-packet separation at the sender. This ratio corresponds to the ratio of available bandwidth and actual bandwidth used by the flow. WTCP tries to achieve the target available bandwidth by keeping this ratio close to one. WTCP uses long-term jitter to track congestion and damp the rate of protocol. However, long-term jitter is not an accurate metric since channel conditions in wireless networks change too rapidly to correlate channel conditions with long-term jitter. In XRTP, we take an approach similar to WTCP, but use observations about short-term jitter which provide a closer correlation to the current state of congestion in the network.

Any rate-based protocol is sensitive to the precision of the operating system's internal clock, (i.e., the clock granularity), which determines the maximum transmission rate of the protocol. Current non-realtime operating systems typically provide granularity of 1-10ms, which supports data rates from 12Mbps to 1.2Mbps respectively for packets of size of 1500 bytes. To achieve higher data rates, it is necessary to send multiple packets per clock tick, which may decrease the benefits gained form using a rate-based protocol. We are currently investigating the effects of such packet streams on XRTP's congestion avoidance and loss discrimination mechanisms.

B. Bandwidth Estimation

The goal of a well-behaved transport protocol is to target its rate at a level close to the available bandwidth in the network. In general, the available bandwidth, and so the target rate, to a flow is a dynamic value, constantly changing with channel usage. Transport protocols can monitor the transmission stream to infer information about bandwidth in the network. However, such information may track two very different parameters: bottleneck bandwidth and available bandwidth. Bottleneck bandwidth can be defined as the maximum throughput that can be obtained between two hosts in the absence of any cross traffic. Available bandwidth is a flow's fair share of the bandwidth in the presence of other flows.

Ideally, transport protocols use estimates of available bandwidth to prevent unfair use. However, measuring available bandwidth or fair share is a non-trivial task in the current Internet where most routers use a first-come first-serve (FCFS) scheduling policy. Keshav [15] observed the intrinsic difficulty in measuring available bandwidth in FCFS routers when looking at the use of packet-pair in congestion control. Essentially, packet-pair always measures bottleneck bandwidth instead of available bandwidth. Protocols like Streaming Media Congestion Control (SMCC) [16] use packet-pair to determine an estimate of available bandwidth. However, their approach assumes the use of RED [17] enabled routers that are not common in today's Internet. Dovrolis [18] showed that available bandwidth could be estimated for FCFS routers in a one hop network using packet-pair along with an estimate of cross-traffic utilization of the bottleneck link. By extending this to an infrastructure-based wireless network with the last hop to the mobile node as the bottleneck link, XRTP estimates the available bandwidth using packet-pair in conjunction with jitter measurements (which indicate cross-traffic utilization) and packet loss. This approach is limited by the number of jitter measurements observed in a round-trip time. Thus, XRTP can use packet-pair to estimate available bandwidth, provided it is able to maintain sufficient packets in flight to gauge network conditions. We are currently investigating how XRTP should behave when a sufficient number of jitter measurements is not received.

The first step to estimating available bandwidth is to accurately determine the bottleneck bandwidth. Techniques like packet-pair have been widely used by off-line tools that measure bandwidth (e.g., *tcpanaly* [19], *bprobe* [20]). However, measurements from packet-pair often include erroneous values and the data filtering techniques used by these tools lack statistical robustness. Lai et. al. [21] suggest the use of kernel density estimation (KDE) to filter data for estimating the bandwidth of all hops from a source to a destination. The disadvantage of directly incorporating such estimation algorithms from these bandwidth measurement tools into a transport protocol lies in the computational complexity of the algorithms. Essentially, offline measurement tools can afford time and resource intensive computations to converge onto accurate estimates, while transport protocols need to be fast and efficient to maximize performance. XRTP uses an optimized KDE algorithm that provides efficient filtering without loss of accuracy using techniques like finite differencing.

C. Loss Discrimination

Solutions for adapting transport protocols for networks with heterogeneous loss characteristics can be classified into three approaches: transparent or infrastructure-based, hybrid or infrastructure-assisted and end-to-end. Infrastructure-based approaches try to hide losses from the transport protocol, which is most often TCP, by adding changes along the path, either at the link layer [22] (i.e., TCP-SNOOP [2]) or at the transport-layer (i.e., indirect-TCP [3]). Such solutions either blindly recover all losses, which may negatively impact some protocols, or require explicit knowledge of the transport protocol, which may not be extendable to new transport protocols. Additionally, infrastructure-based approaches like I-TCP violate the end-to-end semantics of TCP by allowing an intermediate node to acknowledge data that has not yet been acknowledged by the receiver. Finally, all infrastructure-based approaches require the extensive deployment of these intermediate nodes, most likely at the base stations, which limits the feasibility of deployment.

In infrastructure-assisted or hybrid approaches, the transport protocol is modified along with support from the underlying network to indicate congestion losses (explicit congestion notification (ECN)) [7] or transmission losses (explicit loss notification (ELN)) [6]. Krishnan et. al. [5] have extensively analyzed the potential performance benefits of using ELN. While such protocols approach ideal behavior due to the accurate classification of losses, deployment of such hybrid approaches is limited by the necessity to change routers in the network.

The final approach tries to classify losses using purely end-to-end mechanisms (i.e., no support from or changes to the infrastructure) [5], [23], [24], [25]. While end-to-end approaches require changes to the end hosts, deployment can be incremental, falling back on traditional TCP if one of the end hosts does not support the enhanced protocols. A number of protocols have been proposed that enhance TCP to achieve loss discrimination. TCP Westwood [4] does not have an explicit scheme to discriminate losses, but the authors claim that rate estimation can account for wireless losses. Bandwidth estimates are based on information about the rate at which ACKs are received. After a packet loss indication (i.e., three duplicate ACKs), which could be due to either congestion or link errors, the sender uses the estimated bandwidth to properly set the congestion window and the slowstart threshold. However, this scheme does not accurately account for all types of losses making it too conservative to discriminate all transmission losses. Biaz et. al. [23] distinguish between congestion and wireless losses using packet inter-arrival times at the receiver. Barman et. al. [24] assume that the variation in the round trip time and the nature of the loss are correlated. Essentially, congestion losses cause the RTT to vary over the standard deviation but random losses do not. Samaraweera [25] correlates the round trip time with the throughput-load graph of the flow. They assume that a packet loss about a certain threshold is usually due to congestion, otherwise it is due to wireless or random loss. The limitations of all these end-to-end approaches are due to the fact that they propose their heuristics for TCP-like ACK-clocked protocols that are bursty in nature. However, these heuristics are better suited for rate-based protocols that transmit at regular intervals [13], providing better correlation of actual network conditions with heuristic parameters like round-trip time, variance of round-trip time or the throughput-load graph of the flow. In the context of rate-based flow control, XRTP can correlate inter-packet spacing or measured jitter with network conditions to discriminate congestion-based losses from transmission-based losses. To be conservative in discrimination and reduce misclassification of losses, XRTP also incorporates the relative one-way trip time and its variance into the heuristic to better gauge network conditions.

D. Congestion Avoidance

While bandwidth estimation provides a target transmission rate for transport protocols, the estimates may be too high, resulting in congestion in the network. Therefore, all transport protocols must be able to react to an indication of congestion (i.e., congestion control). However, it may be beneficial for the protocol to react to pending congestion with the goal of alleviating the problem before it gets too severe (i.e., congestion avoidance). Although it is certainly beneficial to avoid congestion, this benefit might come at the expense of reduced throughput even in the absence of congestion in the network. Therefore, congestion avoidance plays a crucial role in the ideal behavior expected from transport protocols.

One of the first transport protocols to use congestion avoidance was TCP Vegas [26], which tracks congestion in the network by looking at the difference between the expected throughput and the actual throughput. The baseline for expected throughput is an estimate of the minimum round trip time between the sender and the receiver. If the difference between the estimated and actual throughput is too small, TCP Vegas infers that it is operating close to the capacity and backs off to avoid causing congestion. Similarly, if the difference is too high, TCP Vegas infers that there is extra capacity in the network and increases its rate. The biggest problem with the congestion avoidance algorithm in TCP Vegas is that it uses constant parameters making it unsuitable for dynamic environments [27]. The result is that TCP Vegas is either too aggressive in some situations or not aggressive enough in others. Additionally, while TCP Vegas works well when only TCP Vegas flows are present in the network, it can get shut out by flows using other TCP variations, including TCP Reno and TCP Newreno. To address these problems, XRTP dynamically adapts its congestion avoidance parameters to ensure effective use of network resources and fairness to competing flows.

While TCP Vegas uses explicit congestion avoidance mechanisms, TCP Westwood [4] uses implicit feedback. TCP Westwood performs an end-to-end estimate of the bandwidth available along a TCP connection by measuring the rate of returning acknowledgments. This estimate is used to determine the transmission rate of the protocol. Unfortunately, this implicit scheme is not reactive enough to changes in dynamic environments including wireless networks.

III. Extended Rate-based Transport Protocol

To support data transmission in combined wired/wireless environments, we present XRTP, a rate-based protocol that supports discrimination between congestion and transmission losses. The rate-based component of XRTP provides smooth transmission of data, while supporting effective estimation of network conditions. The XRTP receiver closely monitors available bandwidth through a combination of packet-pair and measured jitter. This monitoring allows the receiver to react to pending congestion (i.e., congestion avoidance) and classify losses. The bandwidth estimation techniques are enhanced with a non-parametric density estimator to filter out anomalies. In this section, we present the protocol architecture followed by the main mechanisms of XRTP: bandwidth estimation, congestion avoidance and rate control, loss discrimination and reliability.

A. Protocol Architecture

XRTP is targeted at environments where losses caused by transmission errors are orders of magnitude greater than for current wired links. XRTP reduces the transmission rate based on measured positive jitters to avoid network congestion. In a lossy environment, XRTP stabilizes the throughput with a loss discriminating heuristic by breaking the fundamental assumption that all packet losses are due to congestion. The protocol relies on the fact that loss of any type creates an anomaly in the jitter. By classifying the anomaly accurately, XRTP categorizes the packet loss, achieving better throughput compared to protocols without loss discrimination.

The design of XRTP decouples reliability from flow control. By combining rate-controlled acknowledgements with a monotonically increasing sequence number for all packets, XRTP eliminates the need for a retransmission

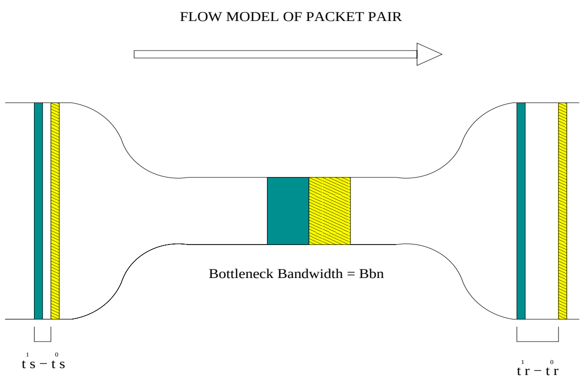

Fig. 1. Fluid model of packet pair with two packets.

timer. Essentially, sequence numbers and gap detection are used to detect losses and window-based selective acknowledgements are used to inform the sender about any losses. While loss discrimination is used to determine the impact of a loss on the transmission rate, the reliability mechanisms (i.e., retransmission) can be based on more application-specific knowledge. For bulk data traffic, XRTP retransmits all lost packets. To support multimedia traffic, XRTP uses information like packet deadlines and priorities to determine which packets should be retransmitted.

B. Bandwidth Estimation

XRTP estimates available bandwidth using packet-pair and jitter measurements at the receiver to set a target rate at which the protocol transmits. Bottleneck bandwidth needs to be estimated constantly since it could vary dynamically due to changes in channel conditions. The advantage of measuring bottleneck bandwidth using packet-pair is due to the fact that it can be done with two data packets and it is very simple to implement. This section describes the packet-pair technique and the statistical method used to filter the observed data to obtain accurate estimates.

1) Packet Pair: The fluid flow model in Figure 1 depicts two packets of the same size traveling from a source to a destination. The narrow part represents the bottleneck link. Once the packet-pair leaves the bottleneck link, a gap is inserted between the two packets due to queuing at the bottleneck link. Let B$_{bn}$ be the bottleneck bandwidth, S be the size of the packet, t_0^s and t_1^s be the time when the first and second packets are sent back-to-back from the sender and t_0^r and t_1^r be the time when they are received at the receiver. To maintain conservation of flow, the equilibrium equation for the described model is given by

$$t_1^r - t_0^r = max\left(\frac{S}{B_{bn}}, t_1^s - t_0^s\right). \tag{1}$$

This equation reveals that if the two packets are sent close enough in time forcing them to queue at the bottleneck link back-to-back, the packets arrive at the receiver with the same spacing, $t_1^r - t_0^r$, as the spacing introduced by the bottleneck link's bandwidth, $\frac{S}{B_{bn}}$. Rearranging Equation (1), the bottleneck bandwidth can now be computed as

$$B_{bn} = \frac{S}{t_1^r - t_0^r}. \tag{2}$$

However, Equation (2) only holds provided that the two packets are queued only at the bottleneck link and at no later link downstream, the routers implement first-come first-serve (FCFS) queuing with simple store and forward and the transmission delay is linear with respect to packet size [21]. Among these assumptions, it is extremely difficult to satisfy the first since packets travel through multiple hops and each hop carries multiple flows. Due to this dynamic nature of the network, packets from other flows can get inserted between the packet pairs increasing the gap between the packets and causing an underestimation of the bottleneck bandwidth (time-expansion). The packet-pair may also get queued at a later router decreasing the gap between packets causing an overestimation of bottleneck bandwidth (time-compression).

```
array h[n] // last n window-width
array obsrv[n] // last n observations
array pdf[n] // density estimations
est // current estimate
function bandwidth_estimation(new)
   // subtract the density of the oldest observation
   for i = 2 to n
      $pdf[i] \; -= \frac{K(\frac{est-obsrv[i]}{h[0]})}{n.h[0]}$
   next i
   // delete the oldest observation along
   // with addition of new density estimate
   for i = 1 to n-1
      $pdf[i] \; = pdf[i+1] + \frac{K(\frac{est-new}{h[n]})}{n.h[n]}$
      obsrv[i] = obsrv[i+1]
      h[i] = h[i+1]
   next i
   // compute the density of new observation
   obsrv[n] = new
   pdf[n] = $\frac{1}{nh[n]} \sum_{i=1}^{n} K\left(\frac{est-obsrv[i]}{h[n]}\right)$
    // Compute new bandwidth i.e., compute mean
   $h[n] = \frac{mean}{10}$
   // return new estimate
 end function
```

Fig. 2. Linear kernel density estimation for BW filtering.

To satisfy the first assumption and enable the use of packet-pair for bottleneck bandwidth estimation, a statistically robust filtering algorithm is necessary to eliminate anomalies like time-compression and time-expansion. XRTP uses a modified version of a kernel density estimation (KDE) algorithm [9] particularly adapted to the rate-based protocol.

2) Kernel Density Estimation: Probability density functions are the basis for estimation of any random quantity. Consider the set of observed data points of an unknown probability density function. To avoid making assumptions about the distribution of the observed data, a non-parametric approach such as KDE [9] can be used to filter the observations. The best estimate of the observed data would be the mean of this random variable.

The probability density function for a kernel estimator with kernel K is defined by

$$f(x) = \frac{1}{nh(x)} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h(x)}\right). \tag{3}$$

Here, *h(x)* is the variable window width, also called the smoothing parameter or bandwidth, *n* is the number of observations collected and *K* is the kernel function that satisfies the following condition

$$\int_{-\infty}^{\infty} K(x)dx = 1.$$

The time complexity of the KDE algorithm is of order $O(n^2)$. Unlike Lai et. al. [21] who used this estimation technique for measuring bottleneck bandwidth in their off-line tool *pathchar*, estimating bandwidth on-line in a transport protocol requires that the estimator algorithm be optimized to work as efficiently as possible. Given the estimator function as described in Equation (3), for a fixed width *h*, a standard finite differencing technique can be used to reduce the algorithm to linear time (see Figure 2). Specifically, each time a new observation or measurement is added into the estimator, the density of the observation being removed (oldest observed measurement) can be subtracted from all observations and the density of the new observation can be added to all observations in time linear in n. Therefore, XRTP uses this linearized KDE algorithm to filter out irrelevant observations (i.e., effects of time-compression and time-expansion) to accurately estimate bottleneck bandwidth.

C. Congestion Avoidance and Rate Control

When there are multiple flows sharing a link, running XRTP at the bottleneck rate would cause congestion since the bottleneck rate is not the fair share. To prevent the potential of congestion by transmitting data at a rate greater than the fair share, XRTP engages explicit congestion avoidance techniques along with congestion control. XRTP uses three mechanism to regulate its rate. First, XRTP integrates new bandwidth estimates, which slowly push XRTP to the rate of the bottleneck bandwidth. This is essentially XRTP's rate increase mechanism. Second, XRTP uses jitter measurements to determine if the current rate is higher than XRTP's fair share, in which case the rate is pushed down. Finally, XRTP cuts its rate by one half when it detects a congestion loss, providing multiplicative decrease. Figure 3 summarizes the events that impact the rate of transmission of a XRTP sender, which we describe in detail in the reminder of this section.

Event	Action	Comments
Bandwidth estimation	$rate_{new} = rate_{current} * \alpha + new_{estimated\_rate} * (1 - \alpha)$	α is dynamically adjusted based on estimates of network conditions (see equations 5 and 6).
Positive jitters	$rate_{new} = rate_{current} - \frac{3}{\sum of\ past\ 3\ positive\ jitters}$	-
Congestion loss	$rate_{new} = \frac{rate_{current}}{2}$	XRTP only cuts its rate to a congestion loss once per RTT.

Fig. 3. List of events and actions taken by XRTP to maintain optimal transmission rate.

1) Using bandwidth estimations: As a first estimate of transmission rate, XRTP sends packet-pairs to estimate the bottleneck bandwidth. Each time the receiver estimates the new bandwidth, it uses a weighted average of the current rate and the new estimated bandwidth to update the rate of the protocol. Exponentially weighted moving average (EWMA) is a poor estimator to measure bottleneck bandwidth accurately since it cannot eliminate the spurious observations that KDE eliminates elegantly. However, EWMA is extremely efficient in maintaining moving averages provided the observations are accurate. Therefore, the XRTP sender uses the new KDE filter to accurately estimate bottleneck bandwidth to update the new rate using the standard EWMA equation with a smoothing parameter α (see Equation 4). XRTP uses Equation 4 to update its rate once every k acknowledgments, where k is set to 3. Setting k to any reasonable value ($3 \leq k \leq 10$) has little impact on the performance of the protocol as long as it keeps XRTP reactive to changing bandwidth within a single RTT. Setting k to extremely large values makes XRTP unresponsive and too conservative.

$$rate_{new} = rate_{current} * \alpha + new_{estimated\_rate} * (1 - \alpha) \tag{4}$$

Left alone, this estimation tracks the bottleneck bandwidth, which cannot be used alone to determine the transmission rate since other flows might be sharing the network. However, this estimation can be combined with two novel congestion avoidance techniques, *optimizing XRTP's parameter, α, based on network conditions* and *regulating the transmission rate based on jitter*, to effectively operate at the network's fair share.

2) Optimizing α based on network conditions: To compete for unused bandwidth, a transport protocol must probe the network. Similar to TCP's use of additive increase, XRTP increases its rate in the absence of congestion. Rate increases in XRTP are governed by the inclusion of new bandwidth estimates, as shown in Equation 4. Since these estimates are measured using packet pair, they provide an estimate of the bottleneck bandwidth, not the available bandwidth. Therefore, XRTP uses this estimate as a target for probing the network. The rate at which XRTP tries to reach this target is determined by α, the weight of the new measurement.

In general, α can range from α_{min} to α_{max}, where α_{min} is the most aggressive and α_{max} is the least aggressive. To be conservative at the start, α is set equal to α_{max}. When XRTP suffers no packet losses in two consecutive RTTs, α is driven towards α_{min}, allowing it to quickly react to any available bandwidth using Equation 5.

$$\alpha_{new} = \frac{\alpha_{current} + \alpha_{min}}{2} \tag{5}$$

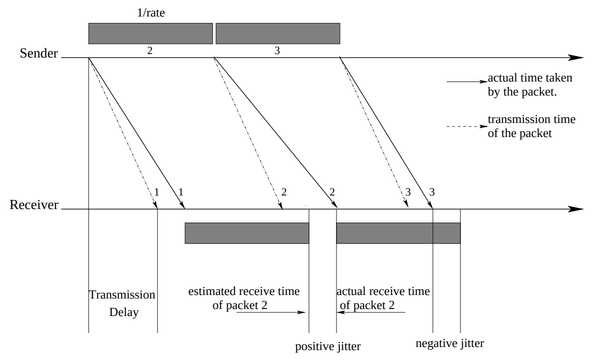
1/rate
Sender
2
3
actual time taken by the packet.
transmission time of the packet
Receiver
1
1
2
2
3
3
Transmission Delay
estimated receive time of packet 2
actual receive time of packet 2
positive jitter
negative jitter

Fig. 4. Jitter caused by variation in queuing delay.

When XRTP suffers multiple packet losses due to congestion in a single RTT, the protocol pushes α closer to α_{max} to prevent the rate of transmission from ramping up too quickly using Equation 6.

$$\alpha_{new} = \frac{\alpha_{current} + \alpha_{max}}{2} \tag{6}$$

XRTP is conservative and waits for indications from the network maintaining α at the current level when there is no packet loss in a single RTT.

The novelty of XRTP's congestion avoidance algorithm comes from the fact that XRTP adapts α dynamically based on the state of the network. Other protocols that use congestion avoidance, such as TCP Vegas, use fixed parameters, which either result in a protocol that is too aggressive and adds to congestion in the network or is not aggressive enough to compete with other flows. During normal XRTP operation, α converges to a lower value, allowing XRTP to compete aggressively with other flows that are also probing the network. However, if there is congestion in the network, α, converges to a higher value, assuring that XRTP is not too aggressive and does not add to the congestion.

3) Regulating rate based on jitter: While probing is necessary to find the unused bandwidth in the network, a good congestion avoidance algorithm must also adapt its rate down if it is sending at a rate higher than its fair share and causing congestion. To support such congestion avoidance, XRTP monitors for pending congestion by monitoring jitter in the transmission stream. Since XRTP also sends packets at regular intervals, they are expected to be received at regular intervals; in other words, the inter-arrival time between packets at the receiver should be equal to the inter-sending time at the sender. In reality, the inter-sending time may not equal the inter-arrival time at the receiver due to queuing delays and the presence of other flows (See Figure 4). XRTP keeps track of this jitter, the difference between the inter-arrival time and the inter-sending time, to damp the rate and operate close to its fair share of the bandwidth.

To estimate fair share at the receiver, XRTP observes the arrival time of packets. Let the arrival time of the n^{th} packet, t_r^n, be the sum of the time it was sent, t_n^s, and the transmission time, tt_n (See Equation (7)).

$$t_n^r = t_n^s + tt_n \tag{7}$$

Transmission time can be divided into two components, propagation time, pt_n and queue time, q_n (at the routers), as shown in Equation (8). If the routes are not changing, propagation time, pt_n, is constant since it only depends on the transmission speed of the media. Queue time, q_n, mainly includes the time the packets wait to be processed and negligible processing time.

$$tt_n = pt_n + q_n \tag{8}$$

The number of packets along the path of communication represents the load in the network. As this number increases, a packet in transmission experiences longer wait times at the router, and so longer end-to-end transmission times. On the other hand, as network load decreases, end-to-end transmission time decreases to the limits of the propagation delay on the transmission medium. The inter-arrival time, iat_n, between two successive packets,

$n-1$ and n, can be used to determine the presence of congestion in the network. Substituting Equation (7) into Equation (9), we get Equation (10) for inter-arrival time.

$$iat_n = t_n^r - t_{n-1}^r \tag{9}$$

$$iat_n = (tt_n - tt_{n-1}) + (t_n^s - t_{n-1}^s) \tag{10}$$

Since the time the packets were sent is known, it can be subtracted from Equation (10) to get a measure of load in the network as shown in Equation (11). Thus, jitter is governed by the difference in packet travel time. Along with the assumption that propagation time is constant, the only variant is the waiting time in the queue, as shown in Equation (13). Therefore, the main component of jitter is the time the packet spent waiting in the router queues, which in turn indicates the amount of congestion in the network.

$$jitter_n = (tt_n - tt_{n-1}) \tag{11}$$

$$jitter_n = (pt_n - pt_{n-1}) + (q_n - q_{n-1}) \tag{12}$$

$$jitter_n = q_n - q_{n-1} \tag{13}$$

XRTP uses timestamps to determine the difference in send time. Therefore, the jitter for the n^{th} packet can be computed as

$$jitter_n = (t_n^r - t_{n-1}^r) - (t_n^s - t_{n-1}^s). \tag{14}$$

In general, jitter may be positive or negative since delays from the previous or current packet may cause the current packet to arrive early or late. Zero jitter indicates that the queue levels at the routers are maintained at a constant level. Positive jitter indicates increasing queue lengths and occurs when the cumulative rate of all flows is greater than the capacity of some link along the path. Negative jitters indicates decreasing queue length and occurs when the network is unloading or when packets are lost due to congestion. In all cases, positive jitter implies that the current rate could cause unfair use of bandwidth. Thus, positive jitter triggers congestion avoidance and causes XRTP to reduce its rate. The jitters indicate the amount of rate XRTP needs to be cut down to reduce the jitter to zero to prevent queue build up. Intuitively, the size of the positive jitter indicates the amount by which XRTP overshot the available bandwidth. To be conservative, XRTP cuts downs its rate based on the average of the history of the past three positive jitters.

$$rate_{new} = rate_{current} - \frac{3}{\sum of\ past\ 3\ positive\ jitters} \tag{15}$$

4) Congestion Control: Apart from decreasing its rate based on jitter and adapting α, XRTP always reacts to congestion losses by cutting its rate in half, essentially using multiplicative decrease, as done by TCP. After a change in rate, it would take at least half the round-trip time for the sender to react and another half for the receiver to notice a change. Therefore, similar to other transport protocols, XRTP cuts its rate by half once per round-trip time (RTT) for any packet loss due to congestion.

D. Loss Discrimination

The main challenge in loss discrimination is that the scheme must be very accurate with a very low rate of misclassification, in particular, classifying a congestion loss as a transmission loss. Each time XRTP discriminates a loss as a congestion loss, it goes into the congestion control phase as described in the previous section. Since XRTP only cuts down its rate once per RTT regardless of the number of losses during that RTT, loss discrimination during the congestion control phase has no effect on the transmission rate. Essentially, a transmission loss classified as a congestion loss puts XRTP into the congestion control phase and renders the outcome of loss discrimination irrelevant for one RTT. While this makes it very important to achieve accurate discrimination, it also limits the negative impact of misclassifications. Therefore, XRTP uses conservative loss discrimination to reduce misclassifications as much as possible while maintaining good congestion control behavior.

XRTP uses the history of jitter measurements to classify losses. As discussed in the previous section, positive jitter indicates incipient congestion in the network. If congestion is not alleviated, the queues overflow and cause a congestion loss. This loss causes packets adjacent to the lost packet in the queue to become closer to each other.

```
function loss_discrimination(packet p)
  if history of positive jitters followed
  by packet loss and negative jitter then
    LOSS = CONGESTION
  else
    // estimate the network congestion
    $p = \frac{curROTT - minROTT}{maxROTT - minROTT}$
    if $p \geq threshold$ and $l1$ out of last
    $n$ ROTT was over deviation then
      LOSS = CONGESTION
    else
      LOSS = TRANSMISSION
    end if
  end if
end function
```

Fig. 5. Hybrid Loss Discrimination heuristics.

Thus, XRTP's receiver would notice negative jitter for the packet received after the congestion loss. However, XRTP notices negative jitters in two situations: when a packet is lost due to congestion or when the network is unloading. Negative jitters occurs after a congestion loss since the next packet takes the queue slot of the dropped packet, reducing the space between the packet before and after the dropped packet. In the case of network unloading, packets from other flows become more infrequent, thus packets arrive more quickly since they experience less queuing time. To differentiate between the causes of negative jitter, XRTP tracks the jitter preceding the loss. Unloading of the network affects multiple packets in a row, each arriving earlier than expected, resulting in multiple negative jitters. However, a congestion loss is likely to be preceded by packets that are experiencing congestion, resulting in positive jitter. Therefore, positive jitter followed by negative jitter indicates a congestion loss. This characterization is conservative since XRTP reacts to a transmission loss during the reduction of network load as if it were a congestion loss, reducing the sending rate.

In last hop wireless networks, a packet lost due to link errors during propagation takes the same amount of transmission time as a packet successfully transmitted. Therefore, the packet following a transmission loss does not experience any significant change in jitter. Hence, a packet loss with no change in the trend of jitters indicates a transmission loss. Similar to other end-to-end loss discrimination schemes, this heuristic with only jitter as a parameter has limitations. Jitter due to the congestion loss of a packet could be masked by the presence of cross traffic flows since slot emptied by the lost packet could be occupied by a packet from a competing flows. This could drastically reduce the noticeable negative jitter at the receiver. Additionally, several lower layer protocols like IEEE 802.11 implement retransmission for packets lost due to transmission error. Retransmission schemes in lower layers would severely skew the jitter metric making it difficult for the heuristic to classify losses accurately.

Classifying congestion losses as transmission losses is detrimental to the network and other flows. To reduce the occurrence of such misclassifications, XRTP's loss discrimination incorporates the relative one-way trip times (ROTT) [28] and the deviation of ROTT [24] into the heuristic (See Figure 5). XRTP determines the ratio of the difference between the current and minimum ROTT to the difference between the maximum and minimum ROTT. The congestion in the network directly correlates to this ratio. XRTP compares this ratio with a threshold to determine the condition of the network when the packet loss occurred. Whenever the heuristic based on jitter classifies a loss as a transmission loss, XRTP uses additional heuristic parameters (i.e., ROTT and its deviation) to confirm the transmission loss.

DATA packet	$global\_sequence\_number$, $reliability\_sequence\_number$, $timestamp, datasize$,DATA
ACK packet	$global\_sequence\_number\_echo$, $CACK, SACK$, $updated\_rate, echo\_timestamp$

Fig. 6. Content of DATA and ACK packet in XRTP.

```
function process_packet(Acknowledgment ACK)
        Process the ack.SACK and update the
        boolean 'acknowledged_by_receiver'
        to reflect if the packet has already
        been received by the receiver.
        if (ack.CACK $\geq$ Sender.expectedACK) then
                // no packet loss
                // delete all the window entries
                // whose RSN $\leq$ ACK.CACK
        else
                // find all packets that have not been
                // acknowledged by the receiver and
                // whose Sender.GSN $\leq$ ACK.GSN$_{echo}$
                // and schedule for re-transmission.
        end if

end function
```

Fig. 7. Algorithm to process the acknowledgments received at the Sender

E. Reliability

Traditional protocols like TCP Newreno use retransmission timers for ensuring reliability of the data transmitted across the network. However, estimating an accurate retransmission timeout (RTO) is a difficult problem [29], [30]. Allman and Paxson showed that there is an intrinsic difficulty in finding optimal parameters to make an accurate estimation for RTO. [29]. Given the difficulties and inaccuracies associated with any scheme of achieving reliability using retransmission timers, XRTP uses logical sequence numbers to achieve reliability, eliminating the need for an RTO. Additionally, XRTP uses rate-controlled acknowledgements to ensure timely feedback from the receiver.

The use of a monotonically increasing packet sequence number allows XRTP to decouple the packet stream from the data stream. Every XRTP packet has a *global_sequence _number (GSN)* that is used for gap-based loss detection. Additionally, every XRTP packet carries a data packet that has a *reliability_sequence_number (RSN)* that allows XRTP to determine which packet was lost. Finally, this dual sequence number space coupled with selective acknowledgements (SACK) [31] supports the recovery of multiple losses per RTT. Figure 6 shows the header information of XRTP's data and acknowledgement packets. The XRTP sender increments *GSN* for every packet sent, but only increments *RSN* for new data packets (i.e., *GSN* keeps track of the packet stream while *RSN* keeps track of the data stream). The receiver echoes the latest *GSN* received along with the next expected RSN. XRTP's sender maintains a log or window of unacknowledged packets. This window's size is generally set to twice the size of the delay-bandwidth product between the sender and the receiver. Each entry in the sender's window has the following contents: *GSN*, *RSN* and a boolean *acknowledged_by_receiver*. Each time an acknowledgement is received, the sender tries to determine if there is a gap (i.e., a packet loss in the packet sequence sent) according to the algorithm described in Figure 7.

To demonstrate XRTP's logical timer (i.e., *GSN*), Figure 8 describes a scenario where consecutive data packets 5 and 6 are lost in the channel. The sender originally sends data packet 5 with $GSN = 16$ and data packet 6 with

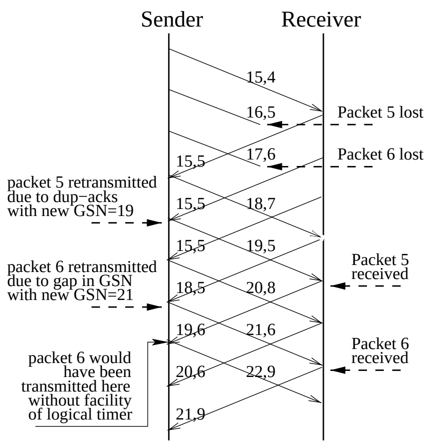
Sender
Receiver
15,4
16,5
Packet 5 lost
17,6
Packet 6 lost
15,5
packet 5 retransmitted due to dup−acks with new GSN=19
15,5
18,7
15,5
19,5
packet 6 retransmitted due to gap in GSN with new GSN=21
Packet 5 received
18,5
20,8
19,6
21,6
packet 6 would have been transmitted here without facility of logical timer
Packet 6 received
20,6
22,9
21,9

Fig. 8. Working of XRTP's reliability without retransmission timers.

$GSN = 17$. Even though duplicate acknowledgements make the sender realize that data packet 5 is lost, it would not know that data packet 6 also is lost. Later, the sender receives an acknowledgement that the receiver is still expecting data packet 5 with an echo of $GSN = 18$. Since the sender was expecting an echo for $GSN = 16$, it immediately knows that data packet 6 is also lost along with data packet 5. The reliability algorithm (See Figure 7) is used to detect the gap and it triggers a retransmission of data packet 5 with $GSN = 20$ and data packet 6 with $GSN = 21$. Thus, every data packet lost can be accurately detected and retransmitted using a logical timer.

XRTP sender's transmission is also limited by the window size. Once the window is filled with unacknowledged packets and the sender receives no new ACKs, the sender assumes that the channel has blacked out and sends one packet every RTT. Similar to the sender, the receiver goes into a blackout mode and sends one ACK every RTT if the receiver does not receive data packets for a threshold amount of time. Note that accurate estimation of RTT is not necessary for XRTP to send a single packet every RTT during this blackout period. XRTP is simply probing the network to keep the connection alive. When the channel conditions improve, the protocol resumes normal transmission.

IV. Performance Evaluation

In terms of performance metrics, the most important goals of any transport protocol are high throughput and fair use of available bandwidth. The evaluations in this section are designed to demonstrate the effectiveness of XRTP's (1) bandwidth estimation, (2) operation in lossy environments, (3) loss discrimination and (4) congestion avoidance. To evaluate XRTP, we use experimental results and simulations in ns-2 [32] (version 2.26). The experiments were conducted over a local wireless area network (IEEE 802.11b between two hosts with available bandwidth of 2Mbps to 11Mbps) to evaluate the KDE algorithm, while the protocol operation was evaluated through simulation.

Figure 9 represents the network topology used for the simulation. Router *R1* has traffic sources using XRTP, TCP, TFRC, TCP Snoop and TCP-Westwood. Link *l2*, from router *R2* to the sink, has a bandwidth of 2Mbps with a 0.01ms propagation delay. This link represents both the wireless link with transmission errors and the shared bottleneck link for all flows in the network. The packet loss rate varies from 0% (ideal case) to 5% to represent a variety of wireless technologies. Link *l1* is set with a propagation delay of 40ms and a bandwidth of 10Mbps. All other links are set with a bandwidth of 10Mbps with propagation delay of 1ms. XRTPs parameters α_{min} and α_{max} are set to 0.95 and 0.99 respectively and α is initialized to α_{max} for all experiments. Each simulation is run for 200 seconds.

A. Comparison of bandwidth estimation using EWMA and Kernel Density Estimation

In this section, we demonstrate the effectiveness of an optimized KDE algorithm to estimate bottleneck bandwidth. Data traces for this experiment were generated by running the packet-pair algorithm on a local wireless area network (IEEE 802.11b) sending packet-pairs every 30 ms. This represents XRTP running on a network with the bottleneck

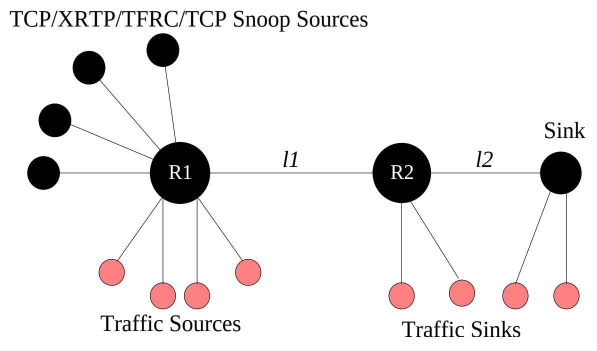

Fig. 9. Infrastructure based wireless network topology setup in ns2.

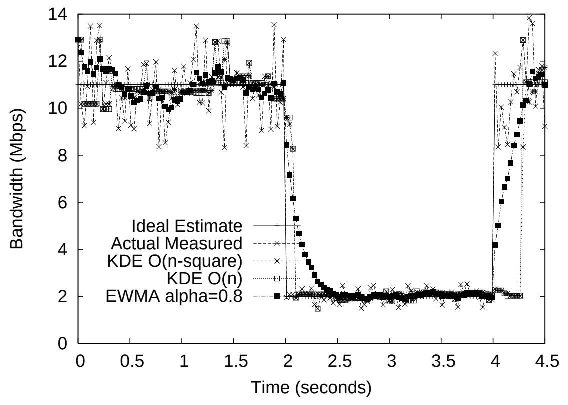

Fig. 10. Bandwidth estimation using KDE and EWMA.

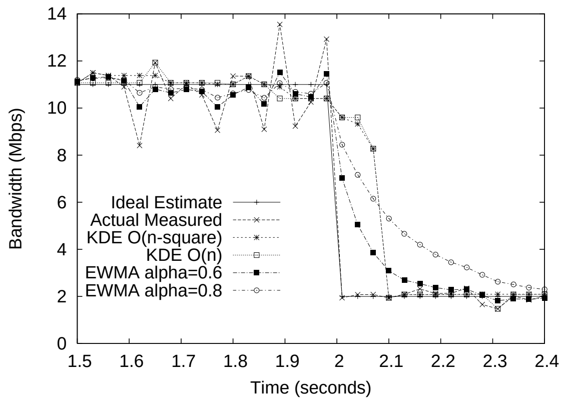

Fig. 11. Snapshot of bandwidth estimation in wireless network.

bandwidth ranging from 2 to 11 Mbps. The receiver was physically moved into regions with different signal strengths to accomplish variations in bandwidth. The traces were used as input to compare EWMA with α set to 0.6 and 0.8 and KDE with time-complexity $O(n^2)$ and $O(n)$, where n is the number of sampled data-points stored to make an estimation. n needs to be determined such that any change in the bandwidth is reflected within a single RTT. For all experiments, n is set to 8, which was found to be appropriate for the latency in the network topology.

Figure 10 represents estimates obtained for these algorithms for one set of generated traces. In this scenario, the bandwidth is 11Mbps from 0 to 5 seconds and falls to 2Mbps for 2 seconds at time 2 seconds. As expected, the algorithms have variance less than the actual measured value (see Figure 10) (i.e., they are less noisy) than the actual observations. The EWMA based estimator takes a longer time to change its estimate from 11Mbps to 2Mbps when the channel bandwidth falls to 2Mbps. Both KDE algorithms are much quicker to adapt to the new 2Mbps bandwidth. To look specifically at the estimated bandwidth traced by the algorithms, a snapshot of Figure 10 from time 1.5 seconds to 2.4 seconds is shown in Figure 11. Notice that EWMA overshoots the channel bandwidth at 1.92 and at 2.01 seconds for α set to 0.6. Both KDE algorithms filter out such spurious estimates and stay close to the channel capacity.

This evaluation shows that KDE performs better than EWMA with agility and efficiency providing accurate estimates of the bottleneck bandwidth by eliminating spurious observations. The KDE $O(n)$ algorithm shows similar estimates compared to the KDE $O(n^2)$ algorithm in terms of accuracy. Selecting larger values of n could provide more accurate estimates of the bandwidth, but at the expense of agility to adapt to fluctuations in channel bandwidth.

B. Evaluation of XRTP's loss discrimination heuristics

The effectiveness of XRTP's loss discrimination techniques is based on the accuracy of the discrimination heuristics. The misclassification of a loss can adversely affect the throughput of the stream or increase congestion in the network. Therefore, we evaluate the probability of each type of misclassification. The goal of the simulations in this subsection is to determine the upper bounds on throughput and the probability of misclassification.

To compare the actual performance gains of using a heuristic to discriminate loss, XRTP was simulated under three different configurations as a single isolated flow on the network topology (see Figure 9). In the first configuration, XRTP was run with no discrimination to gauge the degree of decrease in throughput with the increase in the loss rate over the wireless link (XRTP-ND). In the second configuration, XRTP was run with the perfect discrimination knowledge (i.e., essentially explicit loss notification ELN) to determine the ideal throughput XRTP can provide in a lossy environment (XRTP-PD). In the last configuration, XRTP was run with the loss discrimination heuristic described in Section III. Figure 12 shows a comparison of the performance of XRTP with each configuration along with TCP Newreno, TFRC, TCP Snoop and TCP Westwood, each run separately, as the loss rate in the wireless channel is increased from 0 to 5%. As expected, the throughput of TCP Newreno, TFRC, TCP Westwood and XRTP-ND falls rapidly with increasing loss rate, since every transmission loss is considered a congestion loss causing the protocols to back off. XRTP performs exactly like the configuration with perfect knowledge of the type of loss (XRTP-PD). Note that the two lines for XRTP-PD and XRTP overlap each other in Figure 12. This implies that nearly every transmission loss was accurately discriminated by XRTP.

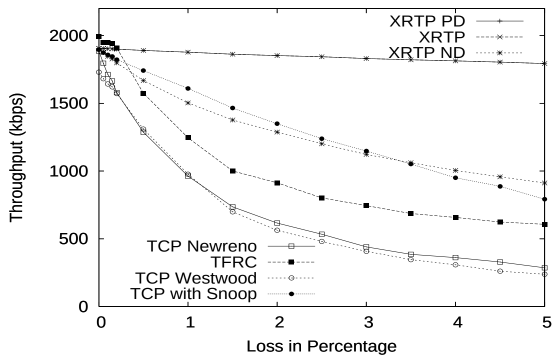

Fig. 12. Performance of XRTP and other flows in isolation.

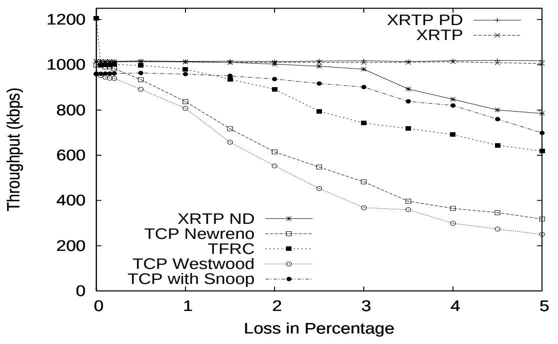

Fig. 13. XRTP and other flows with CBR cross traffic of 1 Mbps.

In the next experiment, XRTP is again run with the same three configurations, but with cross traffic of one CBR flow that uses 1Mbps of the bandwidth on Link *l2* (see Figure 13). As in the previous simulation, TCP Newreno, TFRC, TCP Westwood and XRTP-ND are hit hard by losses due to transmission while XRTP keeps up with XRTP-PD in terms of throughput. To clearly understand how XRTP is always able to maintain bandwidth close to 1Mbps even with increasing channel loss, it is necessary to look at the losses and understand how the protocol discriminated. Table I depicts the performance of the loss discriminator used in XRTP. The first column represents the loss rate on wireless Link *l2* in percentage. Column 2 represents the average of the total number of packet losses across the simulations. Column 3 is losses that did not affect XRTP's congestion control (i.e., losses that occurred during the congestion control phase of XRTP and these losses do not affect the protocol). Columns 4 through 7 represent the losses that occurred when XRTP was not in the congestion control phase. For example, Column 5 is denoted by $C \mid T$ and is read as "*the percentage of total losses that the heuristic discriminated as congestion loss given the loss was actually a transmission loss*". Columns 4 and 5 are loss discriminations that caused XRTP to cut its rate by half. Columns 6 are the losses that XRTP marked as transmission losses when they were congestion losses. Column 7 are losses that were successfully detected by XRTP as transmission losses.

Notice that the number of congestion losses that caused XRTP to reduce its rate is almost constant (the sum of Columns 4 and 5 times Column 2). Thus, the throughput of XRTP remains fairly constant, even with increasing transmission losses. Additionally, as transmission loss rates increase, XRTP misclassifies a larger percentage of transmission losses as congestion losses. However, there were no congestion losses that were classified as transmission losses. The reasons can be attributed to the ideal simulation environment presented by ns2. The bottleneck router *R2* is set with a buffer that is a multiple of twice the delay-bandwidth product of link *l2*. Thus, the CBR flow in the worst case fills only half of the queue since it is using half of the bandwidth. XRTP gets the

TABLE I
LOSS DISCRIMINATION HEURISTICS OF XRTP AGAINST CBR.

1	2	3	4	5	6	7
$loss\%$	$total$	$*$	$C\|C$	$C\|T$	T|C	$T\|T$
0.00	806	82.63	17.36	0.00	0	0.00
0.01	788	82.23	17.63	0.00	0	0.12
0.02	778	81.05	17.99	0.25	0	0.64
0.05	795	82.00	16.10	0.75	0	1.13
0.10	785	79.73	17.83	0.63	0	1.78
0.20	761	77.25	17.87	1.83	0	3.02
0.50	828	75.35	14.13	4.10	0	6.40
1.00	864	68.27	10.41	7.63	0	13.65
2.00	1023	55.11	5.57	10.36	0	28.93
5.00	1346	36.24	1.41	9.50	0	52.82

remaining queue space in the router and all the characteristics required for the heuristic are maintained causing no misclassification of congestion losses. In other scenarios (as shown later), there could be misclassifications of congestion losses as transmission losses, but they are very rare compared to the misclassifications of transmission losses as congestion losses since the parameters of the heuristic are set to conservative values.

To compare XRTP to an infrastructure-based protocol, we included TCP-Snoop in our experiments (See Figures 12 and 13). Clearly, XRTP does better without infrastructure support due to the use of accurate loss discrimination heuristics and selective acknowledgements for reliability. However, it is interesting to consider the impact of enhancing XRTP with a link layer solution like Snoop. TCP Snoop is a last-hop wireless network solution where the base station caches the data packets to improve the performance of TCP. TCP Snoop peeks into the TCP's header to determine the byte sequence number. As expected, TCP Newreno shows significant improvement using Snoop (See Figure 12). Vangala et.al. [33] has shown that TCP Newreno performance improved using TCP Snoop, but TCP SACK's performance deteriorated due to adverse interactions between retransmissions from the sender and from Snoop. This would seem to indicate that XRTP cannot be enhanced by Snoop due to its use of SACK. However, unlike TCP SACK, XRTP would be unaffected by the inclusion of Snoop. If Snoop peeked into XRTP's data or acknowledgement header, it would read the *GSN*, which is monotonically increasing (See Section III-E). Thus, each time a packet lost by transmission in the wireless link is cached at the intermediate link *R2*, the acknowledgement from the receiver for the next succeeding packet after the lost packet invalidates this cache. Therefore, unlike TCP SACK that takes a detrimental hit from Snoop, XRTP gains no benefit from the underlying Snoop protocol. However, Snoop could be updated to work in unison with XRTP, similar to the work in [34], where Snoop was modified to perform better with TCP SACK.

C. Evaluation of XRTP in presence of competing flows and cross traffic

To evaluate the performance of XRTP in the presence of cross traffic, the following experiments were run with the same network topology as described in Figure 9. In the first set of experiments, a single XRTP flow competes against a single TCP Newreno flow with background traffic of CBR flows that cumulatively use 1Mbps of Link *l2*. Figure 14 depicts the throughput of each flow as the loss rate on the link *l2* increases from 0% to 5%. Complementing the graph, Table II depicts the loss discrimination of XRTP.

XRTP clearly shows improved performance due to the use of loss discrimination and improved congestion control. Table II shows that discriminating congestion losses as transmission losses is negligible. It is important to notice that even though the throughput of XRTP is increasing with an increase in loss rate over the wireless link, XRTP is utilizing only that extra bandwidth that is made available by TCP since TCP is reacting to transmission losses as congestion losses (See Figure 14). With increasing loss rates, the percentage of congestion losses (Column 4 and 6) is decreasing, implying that TCP is reacting to transmission losses while XRTP is able to discriminate them as transmission losses. Table II also shows how XRTP actively uses congestion avoidance to prevent packet loss. The total number of packets lost due to congestion rapidly falls with increasing loss rates, implying that XRTP is aware of the other flows in the network.

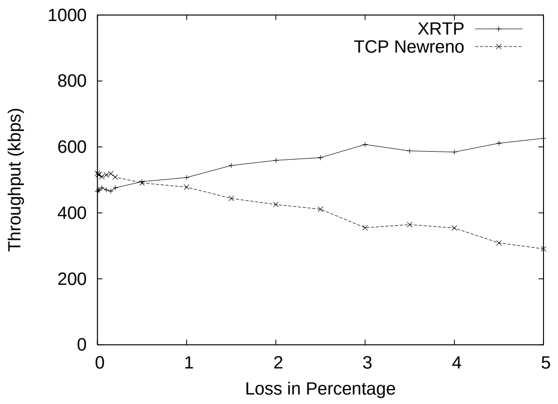

Fig. 14. XRTP versus TCP Newreno with CBR traffic of 1 Mbps.

TABLE II
LOSS DISCRIMINATION HEURISTICS OF XRTP AGAINST TCP WITH CBR

1	2	3	4	5	6	7
$loss\%$	$total$	$*$	$C\|C$	$C\|T$	$T\|C$	$T\|T$
0.00	128	54.68	44.53	0.00	0.78	0.00
0.01	127	51.96	44.09	0.00	3.14	0.78
0.02	144	60.41	36.11	0.69	2.08	0.69
0.05	100	48.00	47.00	2.00	1.00	2.00
0.10	108	42.59	42.59	7.40	2.77	4.62
0.20	93	41.93	35.48	7.52	1.07	13.97
0.50	106	29.23	19.81	16.03	0.00	34.90
1.00	152	7.88	3.94	19.07	0.00	69.07
2.00	255	1.95	0.00	5.88	0.00	92.15
5.00	719	0.82	0.00	1.52	0.00	97.63

Consider the simulation with loss rate of 2%. The total number of losses classified as congestion that affected XRTP is about 15 packets (6% of 255) for the whole 200 second simulation although the throughput averaged around 650Kbps. Comparing it with Table I for the same loss rate of 2%, the total number of losses classified as congestion losses that affected XRTP is about 92 packets (9% of 1023) while the throughput is about 1000Kbps. This confirms that XRTP's congestion avoidance plays an important role in adapting the protocol in the presence of competing flows.

To provide further details about the working of XRTP's pro-active congestion avoidance, Figures 15 and 16 present snapshots of XRTP's throughput and the congestion avoidance parameter α corresponding to experiments in Figure 14 for a loss rate of 0.5% . Both of these figures demonstrate the fact that XRTP reactively adapts to changes in the available bandwidth. In Figure 15, both XRTP and TCP Newreno are sharing the bandwidth fairly. Also, XRTP's throughput dips in Figure 15 corresponding with α peaks in Figure 16. Remember, XRTP is less aggressive when α is closer to α_{max}. Essentially, XRTP is able to detect congestion it is causing in the network and back off appropriately.

1) Congestion Avoidance Behavior: The next experiment explicitly analyzes the congestion avoidance behavior of XRTP as the available bandwidth changes. Here, a single XRTP flow competes against a CBR flow whose rate is incremented by 0.4Mbps every 20 seconds and then decreased at the same rate. Figure 18 represents α as the rate of the CBR flow changes. XRTP starts off with the $\alpha = \alpha_{max} = 0.99$. Since XRTP does not experience any congestion, α drops rapidly to $\alpha_{min} = 0.95$ and averages at 0.95 for the first 80 seconds. α then fluctuates between 0.94 and 0.98 for the next 40 seconds due to the fact that XRTP faced congestion and packet losses between 80-120 seconds, kicking the congestion avoidance mechanism into action. In the last 80 seconds, α moves back to 0.95 since the CBR flows are switching off and more bandwidth is becoming available to XRTP. Corresponding to this timeline, Figure 19 shows the throughput of XRTP against the CBR flow along with the aggregate throughput of both the flows. XRTP satisfactorily tracks the available bandwidth with minimum lag, reacting appropriately to

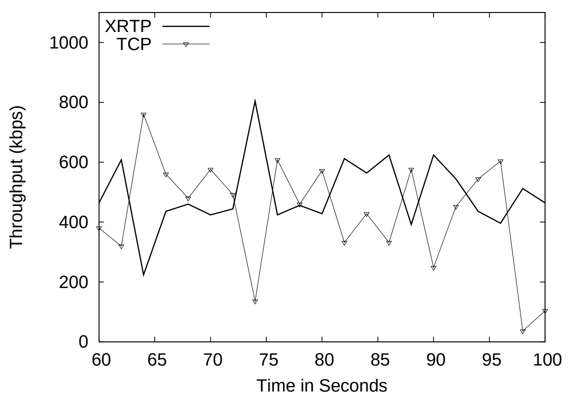

Fig. 15. Throughput timeline of XRTP and TCP against CBR flow of 1Mbps with transmission loss rate of 0.5%

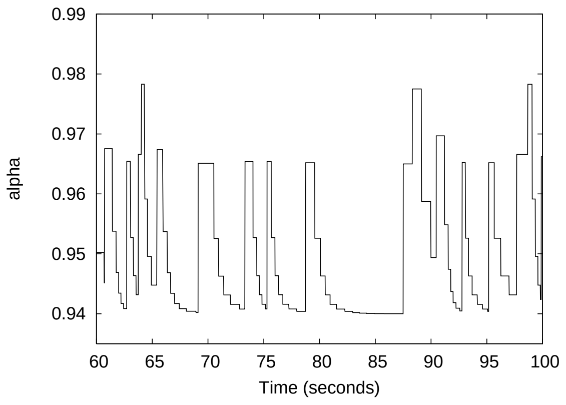

Fig. 16. Variation of XRTP's α versus TCP with CBR flow of 1Mbps and transmission loss rate of 0.5%

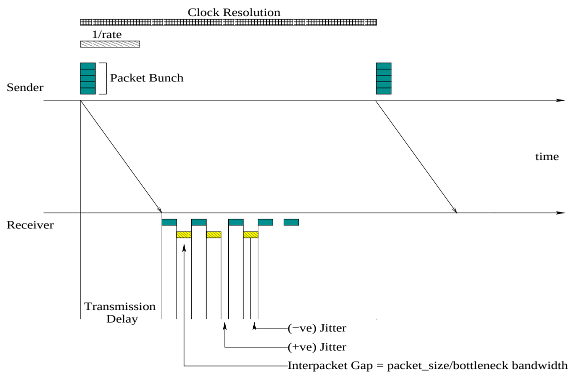

Fig. 17. Jitters caused by variance in propagation delay using coarse grained timer

changes in network conditions.

2) Fairness in the presence of a large number of flows: In the following set of experiments, XRTP's behavior, especially fairness, was evaluated in the presence of a large number of flows. The experiment was carried out on the network topology described in Figure 9. An equal number of XRTP flows and TCP Newreno flows competed against each other for the limited resource link *l2*. Figure 20 presents the cumulative throughput of XRTP against TCP Newreno. The x-axis represents the total number of flows of XRTP and TCP Newreno. Note that the graph shows throughput up to 120 flows since the 2Mbps link is too thin to sustain any more flows. The XRTP and TCP graphs remain close with the increasing number of flows in the network, roughly using half the available bandwidth each. This supports the fact that XRTP flows are able to detect congestion and back off allowing all flows in the network to use the bandwidth fairly. Thus, Figure 20 shows that XRTP is fair to competing TCP Newreno flows in the presence of a large number of flows.

V. Effect of Clock Granularity

XRTP, like other rate-based protocols, is very sensitive to the precision of the operating system's internal clock. For example, on a Linux system running on an Intel platform, the best timer resolution is about 10 milliseconds (called as a jiffy). Assuming packet size of 1000 bytes and timer precision of 10 milliseconds, a rate-based protocol could perform at its optimum only on those network paths whose bottleneck bandwidth is less than 8Mbps. This problem does not arise on simulators like ns2 because of the use of virtual clocks that allow simulators to send packets at the precise virtual time. To handle this situation, a two pronged strategy can be followed: either provide a mechanism in the protocol itself to handle coarse granularity or devise a method to obtain finer granularity in the operating system like KURT Linux [35].

Along the lines of the first strategy, it is obvious that there is a limit to the granularity of the timer beyond which rate-based protocols would start behaving like a TCP protocol. Consider a scenario where a single XRTP

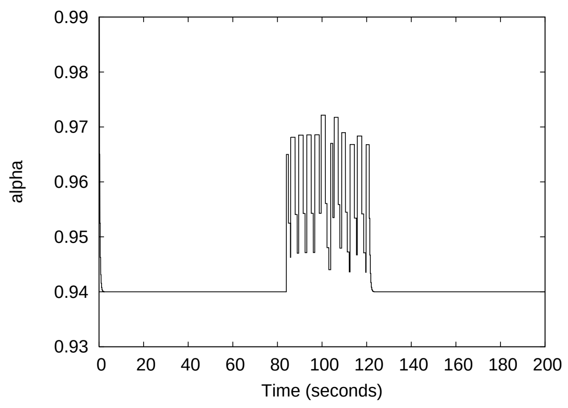

Fig. 18. Snapshot of α against changing available bandwidth

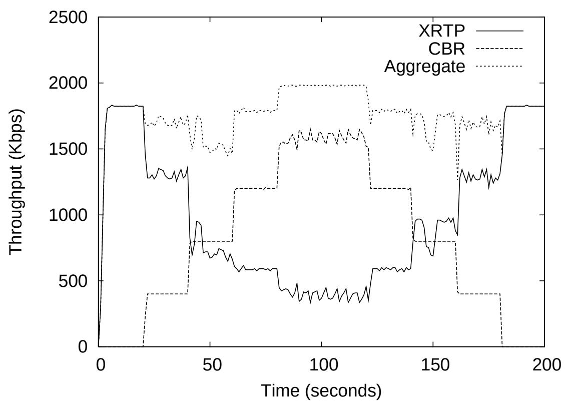

Fig. 19. XRTP's throughput against changing available bandwidth

flow is running on a link which is 100Mbps and the granularity of the clock is 10 millisecond. To obtain optimum throughput, the protocol must send at least a burst of 125 packets in a jiffy with the packet size of 1000 bytes each (as shown in the Figure 17). This would greatly alter the scheme for the computation of jitter and other congestion control parameters of the protocol. Since the time difference between sending two packets could be less that of the transmission time through the bottleneck link, the packets would behave similar to packet-pairs, as described in Figure 1. To accommodate for this factor in the computation of jitter, it is modified as follows:

$$jitter = (t_r^{n+1} - t_r^n) - max\left((t_s^{n+1} - t_s^n), \frac{S}{B_{bn}}\right) \tag{16}$$

With the modification to the computation to jitter, the simulations results of using a coarse-grained rate-controlled protocol has shown results (not included in the paper due to space constraints), similar to the performance of XRTP without loss discrimination heuristic. But the modification to support coarse grained timers with bursty transmission totally undermines the loss discrimination heuristics pushing us towards the direction of devising methods to provide finer granularity in the operating systems for implementing rate based protocols.

VI. Conclusions

In this paper, we propose a rate-based transport protocol for lossy wireless networks that uses bandwidth estimation and jitter measurements to share the medium fairly and accurate loss discrimination to distinguish congestion and transmission losses. Through our evaluations, we show how non-parametric density-based filters can be used efficiently in a transport protocol when compared to noisy EWMA filters. In general, our simulations show that the accuracy of the heuristics are directly tied to the regularity of the flow. Using the effective loss discrimination scheme, XRTP can react wisely to loss, by reducing the sending rate in the presence of congestion, while maintaining the current rate in the case of transmission loss, thereby increasing the throughput of the protocol. This paper has demonstrated the effectiveness of XRTP's congestion avoidance by optimizing the protocols parameters according to network conditions. However, there are still a number of issues we are researching. This paper does not address the effect of coarse-grained timer granularity of current operating systems on the performance of XRTP. We are currently implementing XRTP in the Linux operating system and are evaluating the impact of Linux's coarse grain timers on XRTP. Finally, given our implementations, we plan to evaluate XRTP in realistic scenarios.

References

[1] "IEEE Standard for Wireless LAN-Medium Access Control and Physical Layer Specification, P802.11," 1999.
[2] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *MOBICOM*, 1995.
[3] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *ICDCS*, 1995.
[4] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *MOBICOM*, 2001.
[5] R. Krishnan, M. Allman, C. Partridge, J. P.G. Sterbenz, and W. Ivancic, "Explicit transport error notification (ETEN) for error-prone wireless and satellite networks - summary," in *Earth Science Technology Conference*, 2002.

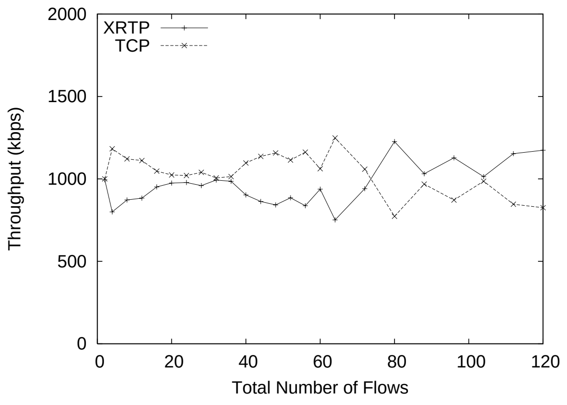

Fig. 20. Fairness of XRTP against TCP Newreno

[6] H. Balakrishnan and R. H. Katz, "Explicit loss notification and wireless web performance," *GLOBECOM*, 1998.

[7] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, 1994.

[8] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM*, 1988.

[9] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, 1986.

[10] R. Rejaie, M. Handley, and D. Estrin, "Rap : An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *INFOCOM*, 1999.

[11] J. Padhye, J. Kurose, D. Towsley, and R. Koodli, "A model based TCP-friendly rate control protocol," in *NOSSDAV*, 1999.

[12] A. C. Feng, A. C. Kapadia, W. C. Feng, and G. G. Belford, "Packet spacing: An enabling mechanism for delivering multimedia content in computational grids," *The Journal of Supercomputing*, vol. 23, 2002.

[13] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," in *INFOCOM*, 2000.

[14] P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "WTCP: A reliable transport protocol for wireless wide-area networks," *Wireless Networks*, vol. 8, 2002.

[15] S. Keshav, "A control-theoretic approach to flow control," *SIGCOMM*, 1995.

[16] N. Aboobaker, D. Chanady, M. Gerla, and M. Y. Sanadidi, "Streaming media congestion control using bandwidth estimation," *IFIP/IEEE International Conference on Management of Multimedia Networks and Services*, 2002.

[17] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, 1993.

[18] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?," in *INFOCOM*, 2001.

[19] V. Paxson, "Automated packet trace analysis of TCP implementations," in *SIGCOMM*, 1997.

[20] R. L. Carter and M. Crovella, "Measuring bottleneck link speed in packet-switched networks.," *Performance Evaluation*, vol. 27/28, 1996.

[21] K. I. Lai, *Measuring the Bandwidth of Packet Switched Network*, Ph.D. thesis, Standford University, 2002.

[22] K. Pentikousis, "TCP in wired-cum-wireless environments," *IEEE Communications Surveys and Tutorials.*, vol. 24, no. 5, 2000.

[23] S. Biaz and N. Vaidya, "Discriminating congestion losses from wireless losses using inter-arrival times at the receiver," in *ASSET*, 1999.

[24] D. Barman and I. Matta, "Effectiveness of loss labeling in improving TCP performance in wired/wireless networks," in *ICNP*, 2002.

[25] N. Samaraweera, "Non-congestion packet loss detection for TCP error recovery using wireless links," in *IEEE Proceedings of Communications*, 1999, vol. 146.

[26] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[27] U. Hengartner, J. Bolliger, and T. Gross, "TCP vegas revisited," in *INFOCOM*, 2000, vol. 3, pp. 1546–1555.

[28] N. Samaraweera and G. Fairhurst, "Explicit loss indication and accurate RTO estimation for TCP error recovery using satellite links," in *In IEE Proceedings - Communications*, 1997.

[29] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *SIGCOMM*. 1999, ACM Press.

[30] V. Paxson, "End-to-end internet packet dynamics," *IEEE/ACM Transactions on Networking*, vol. 7, 1999.

[31] S. Floyd, "Issues of TCP with SACK," Tech. Rep., The ICSI Center for Internet Research, 1996.

[32] VINT Group, "ns2 (network simulator) http://www.isi.edu/nsnam/ns," .

[33] S. Vangala and M.A. Labrador, "Performance of TCP over wireless networks with the snoop protocol," in *27th Annual IEEE Conference on Local Computer Networks*, 2002, pp. 600–601.

[34] S. Vangala and M. A. Labrador, "The TCP SACK-aware snoop protocol for TCP over wireless networks," *IEEE Semiannual Vehicular Technology Conference*, 2003.

[35] Balaji Srinivasan, Shyamalan Pather Robert, Hill Furguan Ansari, and Douglas Niehaus, "A firm real-time system implementation using commercial off-the-shelf hardware and free software," *4th IEEE Symposium on Real-time Technology and Applications,*, 1998.