RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet

Reza Rejaie[*], Mark Handley, Deborah Estrin
University of Southern California
Information Sciences Institute
Marina Del Rey, CA 90292
{*reza, mjh, estrin*}@*isi.edu*

July 15, 1998

Abstract

End-to-end congestion control mechanisms have been critical to the robustness and stability of the Internet. The majority of today's Internet traffic is TCP, and we expect this to remain a large proportion of traffic in the future. Thus, having "TCP-friendly" behavior is crucial for new applications. However, the emergence of non-congestion-controlled realtime applications threatens unfairness to competing TCP traffic and possible congestion collapse.

We present an end-to-end TCP-friendly Rate Adaptation Protocol (RAP), which employs an additive-increase, multiplicative-decrease (AIMD) algorithm. It is well suited for unicast playback of realtime streams and other semi-reliable rate-based applications. Its primary goal is to be fair and in particular *TCP-friendly* while separating network congestion control from application-level reliability.

We evaluate RAP through extensive simulation, and conclude that bandwidth is usually evenly shared between TCP and RAP traffic. Unfairness to TCP traffic is directly determined by how TCP diverges from the AIMD algorithm. Basic RAP behaves in a TCP-friendly fashion in a wide range of likely conditions, but we also devised a fine-grain rate-adaptation mechanism to extend this range further. Finally, we show that deploying RED queue management can result in more ideal fairness between TCP and RAP traffic.

1 Introduction

The Internet has recently been experiencing an explosive growth in the use of audio and video streaming. Such real-time applications are *delay-sensitive*, *semi-reliable* and *rate-based*. Thus they require isochronous processing and quality-of-service(QoS) from the end-to-end point of view. However, today's Internet does not attempt to guarantee an upper bound on end-to-end delay or a lower bound on available bandwidth. As a result, the quality of delivered service to realtime applications over a best-effort network is neither controllable nor predictable. However, lack of support for QoS in the Internet has not prevented rapid growth of realtime streaming applications

[*]This work was supported by DARPA under contract No. DABT63-95-C0095 and DABT63-96-C-0054 as part of SPT and VINT projects

during the last few years and this trend is expected to continue. Many of these applications playback a stored media (i.e. video or audio) for a client over the network. Examples include continuous media servers, digital libraries, distant learning and collaborations, shopping and Entertainment services. These playback clients can afford to slightly delay the playback point and buffer some data to partially absorb variation of the network bandwidth and the end-to-end delay.

In a shared network such as the Internet, all end-systems (both realtime and non-realtime) are expected to react to congestion by adapting their transmission rates, not only to avoid congestion collapse but also to keep the network utilization high. Another important issue is inter-protocol fairness. The rate adjustment should result in a fair share of bandwidth for all the flows that coexist along the same path. Those applications that adapt their transmission rates properly and promptly are known as "good network citizens". Since a dominant portion of today's Internet traffic is TCP-based (e.g. e-mail, FTP, Web-traffic), it is crucial that realtime streams perform TCP-friendly congestion control. By "TCP-friendly", we mean that a realtime flow should obtain approximately the same average bandwidth over timescale of a session as a TCP flow along the same path under the same conditions of delay and packet loss.

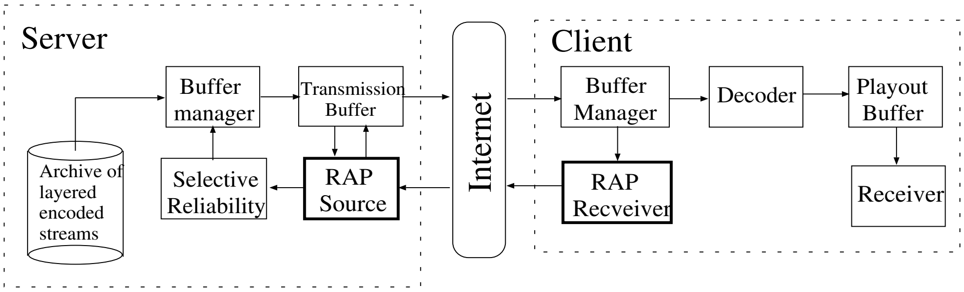


Figure 1: RAP in a typical end-to-end architecture for realtime playback applications in the Internet

We have been working on an architecture for delivery of pre-encoded stored realtime streams over the Internet. Our goal is to make realtime playback applications good network citizens. A typical target application for this architecture could be a web-server or a video-on-demand server that provide access to variety of multimedia streams for large number of users. Figure 1 shows the proposed architecture for server and client. The idea is to separate congestion control from error (quality) control because the former depends on the state of the network while the latter is application specific. The server's transmission rate is continuously adjusted by Rate Adaptation Protocol(RAP) in a TCP-friendly fashion. The RAP module is exclusively in charge of congestion control and loss detection. The quality control module adapts the quality of transmitted streams based on the rate specified by the RAP module. There are many ways to adjust the quality, but the one we are investigating is to use hierarchical (i.e. layered) encoding. The quality control module tries to deliver maximum number of layers that can be fit in available bandwidth. Rate adaptation happens on a timescale of round-trip times (RTT) but layers are added and dropped on a slower timescale by using receiver buffering to accommodate temporary mismatches between transmission and consumption rates. Buffering at the client side also provides the opportunity for selective retransmission as determined by the quality control module. Other types of repair approaches can be deployed if applicable. Note that the aggregate bandwidth used by the server including retransmission should not exceed the bandwidth that is specified by the RAP protocol.

Currently, the majority of realtime applications over the Internet are either open-loop, (i.e. without an end-to-end congestion control mechanism) or non-TCP-friendly. Wide deployment of these applications will have severe negative impact, ranging from unfairness to competing TCP traffic to the potential for congestion collapse. One possible solution would be to make realtime flows use reservation or differentiated service. However, even if such services become widely available, there will remain a significant group of users who are interested in using realtime applications but at low cost. Even in a network that support reservation, different users that fall into the same class of service or share the same reservation still interact as in best effort networks. Thus we believe that congestion control for these applications is critical for the future health of the Internet.

This paper presents design and evaluation of the RAP protocol through simulation. RAP is an end-to-end rate-based congestion control mechanism that is suited for unicast delivery of realtime streams such as the architecture in figure 1 as well as other semi-reliable applications over the Internet. We are also investigating the use of RAP as part of a reliable multicast congestion control scheme. The goals of RAP are to be well-behaved and TCP-friendly.

It has been shown that the Additive Increase and Multiplicative Decrease (AIMD) algorithm efficiently converges to a *fair* state [CJ89]. RAP adopts an AIMD algorithm for rate adaptation to achieve inter-protocol fairness and in particular TCP-friendliness. A rate-based congestion control mechanism is considered TCP-friendly if its *average* transmission rate is not higher (and hopefully not much lower) than that achieved by a TCP connection along the same path [MF97]. RAP performs loss-based rate control and does not rely on any explicit congestion signal from the network. However, if the network supports an explicit congestion signal, RAP could exploit that information to behave more efficiently.

We extensively evaluated performance of RAP through simulation [1]. Our simulations show that RAP is TCP-friendly as long as TCP's congestion control is dominated by the AIMD algorithm. The more TCP's congestion control diverges from AIMD, the less bandwidth is obtained by the TCP traffic. We identified the contribution of TCP's inherent limitations to unfairness of RAP against TCP. Our observations lead us to conclude that RAP behaves in a TCP-friendly fashion over a wide range of scenarios. To further improve RAP, we have also devised a fine grain rate adaptation mechanism that enables it to exhibit TCP-friendly behavior over an even wider range. We assess the impact of TCP's burstiness on inter-protocol fairness in the context of small simulations. Our results show that deploying RED [FJ93] queue management results in an ideal fairness between TCP and RAP traffic. Finally, we investigated self-limiting issues in RAP and did not observe any evidence that implies inherent instability in RAP.

The rest of this paper is organized as follows. We review some of the related work in section 3. In section 4 we present various aspects of the RAP protocol. Detailed description of our simulation results are presented in

[1]Using the ns-2 simulator developed by the VINT project

section 5. Finally, section 6 concludes the paper and addresses some of our future work.

1.1 Rate-based vs Window-based Congestion Control

In Van Jacobson's seminal TCP congestion control paper[Jac88], the idea of using a rate-based approach based on the same mechanisms is mooted. However, for normal reliable transport, window-based solutions offer a number of advantages, including data-driven clocking (avoiding the need for timer-driven transmission) and inherent stability because ACK-clocking means that the number of packets each flow has in the network is kept constant. In practice, this stability property is weakly compromised so that flows can discover the available bandwidth, and because the number of new flows is not limited, but the property is still attractive.

In contrast, rate-based protocols are harder to implement because the correct use of timers is non-trivial and they lack this inherent stability property. However, if we can demonstrate that a rate-based protocol is well behaved, TCP-friendly, and go some way towards addressing this stability issue, then there are a number of applications where a rate-based protocol is a more natural fit than a window-based protocol. For applications that are inherently rate-based, having the congestion control mechanism give them an explicit rate is advantageous. Having the reliability mechanism separated from the congestion control mechanism is a requirement for such semi-reliable applications, and is naturally achieved for rate-based schemes. While this can be done with a separate count of "packets-in-flight" for window-style schemes performing congestion control, the issue of timer-based clocking of new packets when old ones are missing long enough to be assumed to be lost can result in much more bursty behavior, which makes predicting performance for the rate-based application much more difficult. Also, in our reliable multicast congestion control scheme, feedback from representative receivers drives the congestion control scheme, but the representative set changes constantly as network conditions change. Handling these changeovers with a rate-based scheme is much simpler than with a window-based scheme because there is no boundary problem.

None of these reasons is sufficiently strong to rule out window-based mechanisms for these applications, but they were sufficient to challenge us to investigate whether a rate-based approach could be made to co-exist well with TCP. We believe that RAP does satisfy these requirements. RAP is not a radical departure from conventional thinking on congestion control because it is based on the well-understood AIMD algorithm. It is an incremental improvement, and it fills a gap in the current family of protocols that our work on streaming media and reliable multicast has revealed. Our primary contribution is in the analysis of this protocol and its interactions with TCP.

2 Related Work

Congestion control is not a new topic and a large body of work has accumulated describing various congestion control mechanisms. However, the critical work for protocols that wish to co-exist with TCP in the best-effort Internet is somewhat more limited.

Jacob et. al[JE97] propose an architecture for Internet video applications that uses a TCP variant modified so as not to perform retransmission. However, no details of these modifications are given, so it is difficult to tell how these changes affect performance. Normally TCP retransmission control and TCP flow control are closely related - separating these functions probably changes the performance subtly. We discuss in 1.1 why we have chosen a rate based approach rather than a window based approach.

A common approach for rate adaptation is adaptive encoding through the adjustment of codec quantization parameters based on state of the network[OK95]. Many of these studies have not addressed inter-protocol fairness; instead they strive to improve the perceptual quality. However, work in [TZ98] proposes an adaptive coding schemes, using the formula presented in [MF97, MSMO97] that captures the macroscopic behavior of TCP. This approach shows promise, but it has yet to be shown that this formula or the more detailed variant of it in [PFTK98] can be used in a wide range of situations without introducing possible large-scale oscillatory behavior. Moreover, it is CPU-extensive for a server to adaptively encode a potentially large number of streams simultaneously for all active clients.

Cen et. al[CPW98] present the SCP protocol for media streaming in the Internet. The SCP protocol is modified version of TCP that performs TCP Vegas-like rate adjustment in steady state. Their results show that SCP is not TCP-friendly. This may be due to using the shortest RTT that has been measured for rate adjustment since it may widely vary for different flows.

There are many commercial media streaming players that are currently deployed over the Internet such as Realplayer[Net], Vxtreme[Incb] and Microsoft Netshow[Inca]. Although they claim to be adaptive, no analysis is available to verify any claims. Work in [CTCL95] describes VDP protocol that is deployed in Vosaic. Their proposed adaptation algorithm is clearly not TCP-friendly.

Our study differs previous studies of realtime streaming over best-effort networks. We develop a rate adaptation mechanism that will result in inter-protocol fairness and in particular TCP-friendly behavior. The majority of the previous works either did not address fairness at all, or have not examined sufficient cases to attempt to find the bounds where their protocol ceases to be fair.

3 The RAP Protocol

The RAP protocol machinery is mainly implemented at the source. A RAP source sends data packets with sequence numbers, and a RAP sink acknowledges each packet, providing end-to-end feedback. Each acknowledgment(ACK) packet contains the sequence number of the corresponding delivered data packet. Using the feedback, the RAP source can detect losses and sample the round-trip-time (RTT). To design a rate-adaptation mechanism, three issues must be addressed [Jai89]. These are the *decision function* , the *increase/decrease algorithm*, and the *decision frequency*.

Decision Function

A high level description of the rate-adaptation scheme could be summarized by its decision function as follow:

- If no congestion is detected, *periodically* increase the transmission rate;
- If congestion is detected, immediately decrease the transmission rate.

Similar to the congestion avoidance algorithm in TCP [Jac88], the RAP source searches for available bandwidth on the bottleneck link by periodically increasing its transmission rate.

RAP considers losses to be congestion signals, and uses two mechanisms to detect loss: timeouts, and gaps in the sequence space(i.e. ACK-based). These two mechanisms work in parallel.

Similar to TCP, RAP maintains an estimate of RTT, called $SRTT$, and calculates the timeout based on the Jacobson/Karel's algorithm [2]. Nevertheless, it detects the timeout losses differently. Because of the ack-clocking property, a TCP source always receives an ACK and updates the RTT estimate before transmitting a new packet. Thus, the TCP source always has an *updated* estimate of RTT to calculate the timeout. This enables TCP to deploy a timer-driven approach to detect timeout losses. RAP is *not* ack-clocked. A RAP source may send several packets before receiving any new ACK to update the RTT estimate. Therefore, a TCP-like timeout mechanism is not appropriate and results in detecting frequent incorrect losses that are in fact late ACKs.

RAP couples the timer-based loss detection to the packet transmission. The source maintains a record for each transmitted packet, containing the sequence number, departure time, transmission rate and status flag for each packet. The collection of records for outstanding packets is called the *transmission history*. Before sending a new packet, the source checks for a potential timeout among the packets in the history using the updated value of the SRTT estimate. Then it traverses through the transmission history and detects all the timeout losses using the following algorithm:

$$
\begin{array}{l}
WHILE(DepartTime_i + Timeout \geq CurrTime) \\
\quad IF(Flag_i \neq Acked)\ THEN \\
\quad\quad Seq_i\ is\ lost
\end{array}
$$

This mechanism may detect a burst of loss at once. Moreover because of the absence of ack-clocking, the RAP source may still receive a late ACK.

The ACK-based loss detection mechanism in RAP is based on the same intuition as the fast-recovery mechanism in TCP. To limit the amount of overshoot during the increase phase, the RAP source needs to detect congestion (i.e. packet loss) as early as possible. If RAP source receives an ACK that implies delivery of three packets after the missing one, the packet is considered as lost. RAP requires a way to differentiate the loss of an ACK from the loss of the equivalent packet. We have added some redundancy in the ack stream to specify the last hole in delivered sequence number space and provide robustness against single ack losses [3]. An ACK packet contains the following information, as shown in figure 2: 1. The sequence number, A_{curr}, of the packet being acknowledged,

---

[2] $SRTT_{i+1} = \frac{7}{8} SRTT_i + \frac{1}{8} SampleRTT$
$Timeout = \mu * SRTT + \delta * VarRTT$

[3] To achieve resilience to multiple ACK-loss, more information must be carried in ACK packets. We have not studied this issue any further

2. The sequence number, N, of the last packet before A_{curr} that was still missing, or 0 is no packet was missing,
3. The sequence number, A_{last}, of the last packet before N that was received, or 0 is A_{curr} was the first packet.

Using this information, a RAP sender can mark packets as having arrived even when the ACK is dropped because

Packet Loss Pattern	*ACK Information*			
	A_{last}	N	A_{curr}	
1 __	0	0	1	
1 __ __ 4	1	3	4	
1 __ __ 4 __ 6	4	5	6	A_6
1 __ __ 4 __ 6 __ __ 9	6	8	9	
__ 6 __ __ 9 10	6	8	10	
6 __ __ 9 10 11	6	8	11	

Figure 2: ACK-based loss detection in RAP

subsequent ACKs arrive in which their sequence number is greater than N and less than A_{curr}. Adding A_{last} provides sufficient redundancy in the ACK stream to recover loss of ACK_{i-1} when data packet with sequence number i, is lost. After each ACK arrives, the RAP source performs the following algorithm:

$$
\begin{array}{l}
For\ each\ Seq_i\ in\ Trans.\ History \\
DO \\
\quad IF[((A_{curr} \geq Seq_i) AND (Seq_i > N))\ \ OR \\
\qquad (Seq_i = A_{last})]\ THEN \\
\qquad\quad Seq_i\ was\ received \\
\quad ELSE\ IF\ (A_{curr} - Seq_i \geq 3)\ THEN \\
\qquad\quad Seq_i\ is\ lost \\
WHILE(Seq_i \leq A_{curr})
\end{array}
$$

Note that the timeout mechanism is still required as a back up for critical scenarios such as when a burst of packets is lost.

Increase/decrease Algorithm

For its increase/decrease algorithm RAP uses AIMD. In the absence of packet loss, the transmission rate, S, is increased in a step-like fashion. In other words, at each adjusting point we have $S_{i+1} = S_i + \alpha$. We refer to the value of α as *step height*. The transmission rate is controlled by adjusting the inter-packet-gap(IPG). To increase the rate additively, IPG must be iteratively updated based on equation (1) [Jac88]:

$$S_i = \frac{PacketSize}{IPG_i} \qquad IPG_{i+1} = \frac{IPG_i * C}{IPG_i + C} \quad (1), \qquad \alpha = S_{i+1} - S_i = \frac{PacketSize}{C} \quad (2)$$

Note that in the equation (1), C has the dimension of time and it determines the value of α. Upon detecting congestion, the transmission rate is decreased multiplicatively, by increasing the value of IPG [4]:

$$S_{i+1} = \beta S_i, \qquad IPG_{i+1} = IPG_i/\beta \quad \beta = 0.5$$

[4] We use a value of $\beta = 0.5$ which is a conservative choice as it is chosen by TCP (see appendix C in [Jac88]).

Decision Frequency

Decision frequency specifies how often to change the rate. Based on system control theory, optimal adjustment frequency depends on the feedback delay. The feedback delay is the time between changing the rate and detecting the network's reaction to that change. It is suggested that rate-based schemes adjust their rates not more than once per RTT [MF97]. Changing the rate too often results in oscillation whereas infrequent change of rate leads to an unresponsive behavior. Feedback delay in ack-based schemes (e.g. RAP) is equal to one RTT.

RAP adjusts the IPG once every round-trip time. The time between two subsequent adjusting points is called a *step*. Because of the random nature of the RTT signal[Bol93], using the recent sample RTT as the step length is likely to result in a poor behavior. We need a smoothed version of RTT that represents low frequency variation of RTT and filters out the transient (i.e. high frequency) changes. RAP uses the most recent value of SRTT as the step length.

At the beginning of each step, a timer, called step-timer, is set to the recent value of SRTT and the IPG is decreased based on equation (1). The value of IPG remains unchanged until the step-timer expires or a packet loss occurs. If no loss is detected, IPG is decreased and a new step is started. Adjusting the IPG once every SRTT has a nice property; packets sent during one step are likely to be acknowledged during the next step. This allows the RAP source to observe the reaction of the network to the previous adjustment before making a new adjustment.

As we mentioned earlier, C in equation 1 has the dimension of time and is the only parameter that controls the rate of increase of the transmission rate. One immediate question is *what is the right value for C?*. Since our chief goal is to be TCP-friendly, C must be adjusted so that in the steady state, the number of packets transmitted per step is increased by one. Equation 1 allows us to achieve this goal. If the value of IPG is updated once every T seconds and we choose the value of C to be equal to T/k, the number of packets sent during each step is increased by k every step.

RAP uses value of one for k in order to emulate the TCP window adjustment mechanism in the steady state. At each adjusting point, first the step length (i.e. the step timer) and the value of C are set to the recent value of SRTT, and then equation 1 is used to update the value of IPG.

Since the length of each step is SRTT and the height of each step is inversely dependent on SRTT (equation 2), the slope of the transmission rate is inversely related to $SRTT^2$.

$$Slope = \frac{StepHeight}{StepLength} = \frac{\alpha}{SRTT} = \frac{PacketSize}{C * SRTT} \qquad C = SRTT \longrightarrow Slope = \frac{PacketSize}{SRTT^2} \qquad (3)$$

TCP's rate of increase of transmission rate is related to RTT in the same way in the steady state, and so a RAP source can exploit RTT variations and adaptively adjust its transmission rate in the same manner as TCP. The adaptive rate adjustment in RAP is meant to emulate the coarse-grain rate adjustment in TCP. The step length in RAP is analogous to the time it takes for TCP to send a full window worth of packets(i.e. one RTT).

RAP suffers from unfairness against TCP flows with longer RTT in the same way that inter-TCP unfairness has frequently been reported[Flo91]. RAP connections with shorter RTT are more aggressive and achieve a larger share of the bottleneck bandwidth. However, there are two issues to notice: 1. in general, other measures of fairness can only be achieved by implementing the required machinery in the network[She90]; 2. As long as the unfairness problem is not resolved along TCP flows, being TCP-friendly implies accepting this unfairness. Due to lack of space, we have not discussed startup behavior of the RAP protocol. Note that since RAP is designed for relatively long-lived sessions, its behavior during the startup phase is not crucial.

3.1 Clustered Losses

In a shared best-effort networks with a high level of statistical multiplexing, the observed loss pattern has a near random behavior[Bol93] that is determined by the aggregate traffic pattern. Thus it is generally hard for an end system to predict or control the loss rate by adjusting the transmission rate. End systems are expected to react to congestion on suitable time scales by dropping their rate exponentially. It takes one RTT for end systems to detect and react to congestion. Thus an end-system only needs to react at most once per RTT as long as it reacts sufficiently[MF97].

To achieve this end systems require a mechanism by which they can identify a cluster of losses that are potentially related to the same congestion. A simple approach is to ignore all losses that are detected during the first RTT after a back-off.

RAP employs a slightly different approach. If a back-off occurs because packet $Seq_{FirstLoss}$ was lost, the RAP source looks at the sequence number of the last packet that has been transmitted, called $Seq_{LastSent}$. The outstanding packets in the pipe right after the back-off have a sequence number, Seq, in the range:

$$Seq_{LastSent} > Seq > Seq_{FirstLoss}$$

This is one RTT of packets and we call this a *cluster*. Any packet in the cluster can be potentially dropped due to the recent congestion event that was detected by the loss of $Seq_{FirstLoss}$. As the source has already reacted to the congestion, loss of other packets from the cluster are silently ignored. This cluster-loss-mode is triggered by a back-off and terminated as soon as an ACK with sequence number greater or equal to $Seq_{LastSent}$ is received. This mechanism is similar to that employed in TCP-Sack.

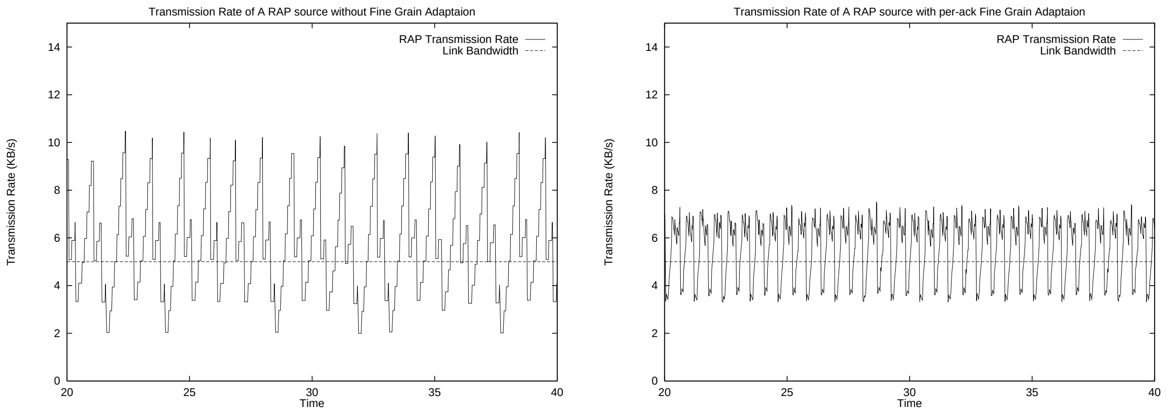
3.2 Fine grain rate adaptation

The AIMD rate adaptation algorithm does not necessarily result in TCP-friendly behavior scenarios when TCP's performance is degraded due to heavy load. Figure 3(a) depicts the transmission rate of a single RAP flow without fine grain adaptation. Note that the rate is monotonically increases until a loss occurs.

The motivation for fine grain rate adaptation is to make RAP more stable and responsive to transient congestion while still performing the AIMD algorithm at a coarser granularity. A fine-grain rate adaptation has two components: 1. Fine-grain feedback: the feedback must only contain a recent fine-grain signal, 2. Fine-grain adjustment: the feedback must only cause minor adjustment in the transmission rate. The long-term mean effect of such adjustment should be zero.

Therefore, we need a fine-grain signal that only captures short-term trends in congestion not captured by packet losses. A short-term exponential moving average of the RTT signal exhibits such a behavior.

However, we require a *dimension-less* zero-mean feedback signal since it is independent of the connection parameters and has a wider applicability. The ratio of the short-term to the long-term exponential moving average [5] of the RTT signal has these desired properties. We have exploited the RTT signal and devised a *continuous* [6] feedback function that is defined as, $Feedback_i = \frac{FRTT_i}{XRTT_i}$, where $FRTT_i$ and $XRTT_i$ are the value of short and long term exponential moving average of RTT samples at the ith adjusting point respectively.



(a) Transmission rate of a single RAP flow without fine grain rate adaptation

(b) Transmission rate of a single RAP flow with per-ack fine grain rate adaptation

Figure 3: Impact of fine grain rate adjustment on RAP's instantaneous transmission rate ($\frac{PktSize}{IPG'}$)

The fine-grain rate adjustment deploys the fine-grain feedback to fine tune the transmission rate. At each *tuning* point, the value of IPG_i is modulated by the feedback signal and the resulting value, IPG'_i, is used for the transmission timer. $IPG'_i = IPG_i * Feedback_i$ (4)

The value of IPG is adjusted iteratively, as we explained earlier, and acts as a *base* transmission rate. Thus, during one step the base transmission rate remains unchanged. However, the actual transmission rate, IPG', adaptively varies with the short-term congestion state. Note that fine-grain feedback does not have an accumulative effect.

Fine-grain rate adaptation could be performed at several granularities, although the feedback signal and the rate adjustment must have the same granularity. We have simulated fine-grain adaptation for two extreme granularities: adapting once per step , and adapting once per ACK. Although both schemes result in improvements, we focused our attention on the per-ack scheme since it has finer granularity, performs slightly better, and is intuitively closer to TCP. We plan to study coarser adaptation schemes by relaxing some restrictions of the per-ack scheme in our

[5] $RTT_{i+1} = (1 - K)RTT_i + K * SampleRTT$

[6] Since our proposed feedback function is continuous, we do not need to deal with specifying thresholds which can be problematic.

future work.

With per-ack adaptation, the arrival of a new ACK causes FRTT and XRTT to be updated, and so hence a new value of IPG' is calculated and applied to the timer. Note that the transmission timer is already running for the next transmission, and so the new IPG' will be used as the next inter-packet gap. Since multiple ACKs may arrive before expiration of the transmission timer, the most recent value of IPG' is used.

The weights for XRTT and FRTT exponential average filters must be chosen so that the feedback signal captures the short-term congestion state since the last ACK. We have set the value of K_{XRTT} and K_{FRTT} to 0.01 and 0.9 respectively. These weights are constant.

Figure 3(b) shows a single RAP flow with per-ack fine grain adaptation. It clearly illustrates that RAP fine tunes its transmission while it follows the AIMD algorithm for coarse grain adaptation.

3.3 Self-limiting issues in RAP

There is a consensus about the inherent instability of rate-based control schemes. Although it is hard to disprove, it has not been theoretically or experimentally proven[MK92].

Self-limiting behavior is the classic problem with rate-based schemes. In window-based schemes the source stops once it has a full window worth of data on the fly. This property makes the window-based schemes intrinsically stable. However, if the source allows retransmission beyond the current window, the stability is lost, and so the number of retransmitted packets must also be limited. Rate-based schemes need to find some variant analogous to the window to bound the volume of outstanding data in the network[Jac97]. One way to achieve this goal is use of correctly implemented timers. In the absence of any feedback, the expired timer forces a source to drop its rate.

RAP achieves self-limiting by using the timeout mechanism for loss detection as a variant of window. In an extreme case when no ACK is received, the transmission rate drops to half once every RTT, until the rate falls below the minimum short-term average rate that the application can tolerate. This worst case scenario only happens if a connection fails (e.g. network is partitioned).

The fine-grain rate adaptation mechanism in RAP effectively strengthens the self-limiting property in RAP and prevents the source from over running the network. During normal operation of a RAP connection in a network with a large number of well-behaved flows, the departure rate of data packets and arrival rate of ACKs are in balance. If the RTT suddenly increases, the arrival rate of ACKs decreases. therefore the number of out-standing packets grows and the balance is lost. If a loss is detected, the rate is dropped to half. Otherwise since the values of new RTT samples is growing, the fine-grain feedback increases[7] the value of IPG' effectively and controls the transmission rate.

[7] specially per-ack fine-grain feedback

3.4 Random Early Discard Gateways

There seems to be general agreement in the community on deploying Random Early Discard (RED)[FJ93] gateways to improve both fairness and performance of TCP traffic. Among other benefits, RED queue management attempts to achieve two goals; it tries to keep the average queue size low, and also by preventing the buffer from filling, it accommodates bursts of packets. One of the main problems for TCP's congestion control is to recover from multiple losses within a window [FF96]. Multiple losses occurs mainly due to buffer overflow in drop-tail queues. Ideally, RED should be configured such that each flow experiences at most one single loss per RTT. Under these circumstances, TCP flows can efficiently recover from a single loss without experiencing a transmission timeout or going through slow-start. Intuitively, as long as a RED gateway operates in its ideal region, RAP and TCP obtain an even share of bandwidth since both use the AIMD algorithm. Nevertheless, if the average queue length exceeds the maximum threshold, RED starts to drop packets with a very high probability. At this point, RAP and TCP start to behave differently. When regular TCP experiences multiple losses within a window, it undergoes a retransmission timeout and its congestion control diverges from the AIMD algorithm. RAP, however, follows the AIMD algorithm as a basis for rate adjustment except for minor tunings due to the fine grain rate adaptation. Regardless of the number of losses within one RTT, RAP reacts only once to the first loss. We expect to observe substantial improvement in fairness by deploying RED even if it only prevents the queue from overflowing. This behavior at least, limits the divergence of TCP's congestion control from the AIMD algorithm in bigger simulations.

Since RED parameters are closely dependent on the behavior of aggregate traffic, it is hard to keep a RED gateway in its ideal region as the behavior of the aggregate traffic changes with time. Thus, configuration of RED is still a research issue.

4 Simulation

In this section we present our simulation results. Our main goal is to explore the properties of RAP, namely TCP-friendliness, ability to cope with background TCP traffic, interaction with RED gateways and the behavior of the fine-grain rate adaptation over a reasonable parameter space. We are particularly interested in RAP's large-scale behavior, i.e. the average performance during a session. Our simulations demonstrate that RAP is TCP-friendly. However in a real network, we would have a larger number of short and long-lived TCP flows with some bursty traffic. Currently, we are in the process of implementing a RAP prototype for actual experiments over the Internet.

We have simulated RAP using the ns2 simulator [MF95], and compared it to TCP Tahoe, Reno, NewReno [FF96] and Sack [MMFR96].

Figure 4 shows the topology of our simulations. The link between SW_1 and SW_2 is always the bottleneck and SW_1 is the bottleneck point. All the other links have higher bandwidth and shorter delay than the bottleneck. The switches implement FIFO scheduling and DropTail queuing except in RED simulations. m RAP connections

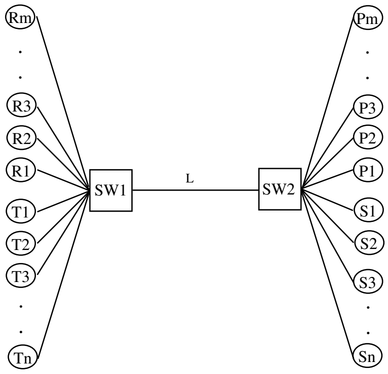


Figure 4: Simulation Topology

from sources $R_i...R_m$ to receivers $P_1...P_m$ share the bottleneck bandwidth with n TCP flows from sources $T_1...T_n$ to receivers $S_1...S_n$. Data and ACK packet sizes are similar for RAP and TCP flows. For a fair comparison, all connections have equal end-to-end delay. The total delay of the side links for each flow is fixed, but is randomly split between $R_i \rightarrow SW_1$ and $SW_2 \rightarrow P_i$ so that the flows are out of phase. The buffer size at SW_1 is four times the RTT-bandwidth product of the bottleneck link, except where otherwise stated.

All simulations were run until they exhibited steady state behavior. All TCP flows are "FTP" sessions with an infinite amount of data. The TCP receiver window is large enough that it is not a limitation. TCP and RAP sources start in a random order with a uniform random delay between their start times. This random delay lessens the resonation between sources and reduces the duration of the initial transition phase. Simulation parameters are summarized in table 1.

Packet Size	100 Byte	Tot. Delay of Side-Links	6 ms
ACK Size	40 Byte	TCP Timeout granularity	100 ms
Bottleneck Delay	20 ms	CP Maximum Window	1000
Bottleneck Buffers	4 * bottleneck B/W * RTT	B/W per flow	5 KByte/s
B/W of Side Links	1.25 MByte/s	Simulation Length	120 sec

Table 1

The average goodput for each flow is measured by the number of delivered packets[8] during the last three quarters of the simulation time to ignore transient startup behavior. Since loss only occurs at the bottleneck switch, the average goodput represents the average bottleneck bandwidth share for each flow. Typically we will graph the mean, min and max bandwidths of the flows of each protocol to examine fairness and identify phase effects.

We initially observed severe phase effect phenomena among in our simulations. Phase effects becomes more pronounced as the number of flows and the amount of resources (i.e. buffer size and bandwidth) increase. This

[8]Here we implicitly assumed that number of duplicate packets a TCP flow receives is negligible and our experiments confirm this assumption.

occurs mainly because of Drop Tail gateways in our small deterministic network as was reported in [FJ92]. Moreover, in our simulations all flows have the same packet size and observe similar RTT, which increases the probability of phase effects.

To eliminate this problem without changing our parameters, we have added a small uniform random delay before transmission of each TCP packet[9]. This delay ranges from zero to the bottleneck service time and emulates the random packet-processing time of intermediate gateways [FJ92]. Obviously, adding this random delay slightly decreases the transmission rate of TCP because it always delayed the transmission. We have added similar randomness to RAP, not only to resolve the phase effect problem between RAP flows, but also compensate for the added delay to TCP flows. Each RAP source adds a random delay ranges from zero to the bottleneck service time, to the value of IPG' before scheduling transmission of the next packet[10]. In a real network, adding randomness is more crucial for RAP than TCP. Because of the ack-clocking and random change in RTT, TCP experiences some randomness. Since RAP is not ack-clocked, the RAP source needs to slightly randomize the IPG' to resolve the phase effect problem. Our fine-grain feedback seems to achieve this goal. Another solution to the phase effect problem is to replace the DropTail gateway with a RED gateway. We have explored this in our simulations as we report later. We believe that the phase effect problem deserves more attention. We plan to investigate the phase effect among RAP flows as well as phase effect between TCP and RAP flows in our future work.

4.1 Evaluation Methodology

In an environment with large numbers of parameters, it is generally hard to isolate a particular variable and study its relation with a particular parameter because of existing inter-dependency among variables. In particular, TCP's behavior changes drastically with configuration parameters and it has some internal constraints. Therefore, it is crucial to distinguish an effect that is caused by TCP's performance constraints from those phenomena that are due to coexisting RAP flows. During our simulations, we attempted to minimize this problem by considering the following guidelines:

1. To identify the impact of TCP's constraints from the inter-protocol dynamics on our results, we have compared RAP with different flavors of TCP.

2. We limited the side-effect of resource (i.e. bottleneck bandwidth and buffer space) contention by scaling up resources proportional to the number of flows so that the amount of resource share per flow remains fixed across simulations. Since the bandwidth and the buffer size of the bottleneck link are scaled up with the same rate, the maximum queuing delay does not change across simulations. The impact of resource contention is also studied separately.

[9] This is possible by using *overhead* configuration parameter of TCP agent in ns.

[10] Although adding the randomness substantially lessens the phase effect problem, the problem still occurs occasionally in big simulations.

3. We chose configuration parameters so that the TCP congestion window tends to be sufficiently large that TCP remains in its well-behaved mode.

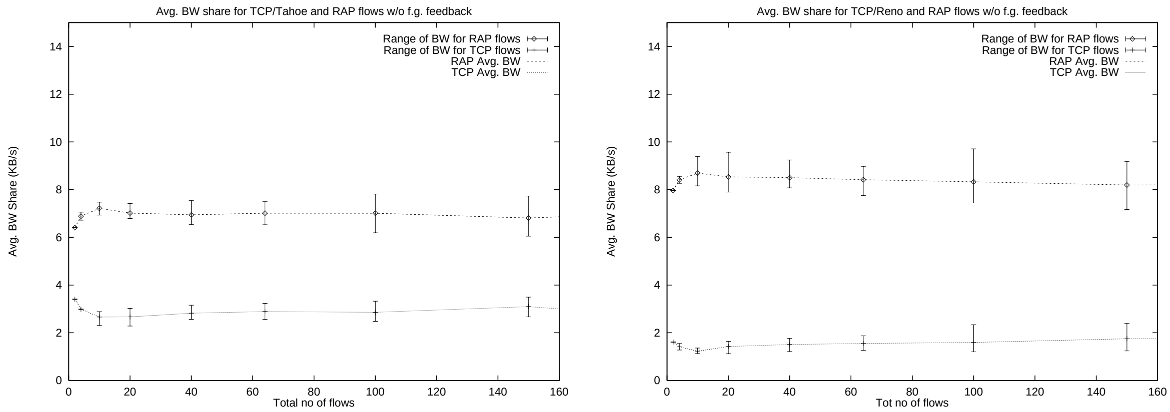
4. We have explored a reasonable portion of the parameter space to examine RAP's behavior over a wide range of circumstances.

5. As a baseline for comparison, we occasionally replaced all the RAP flows with TCP and ran the same scenario without any RAP flow. We call this *TCP base-case*. The TCP base case may help us to separate phenomenon that are related to TCP traffic.

4.2 Experiments and Results

4.2.1 TCP-friendliness

The first set of simulations examines the TCP-friendliness of RAP without fine-grain rate adaptation. Figure 5(a)



(a) RAP coexisting with Tahoe

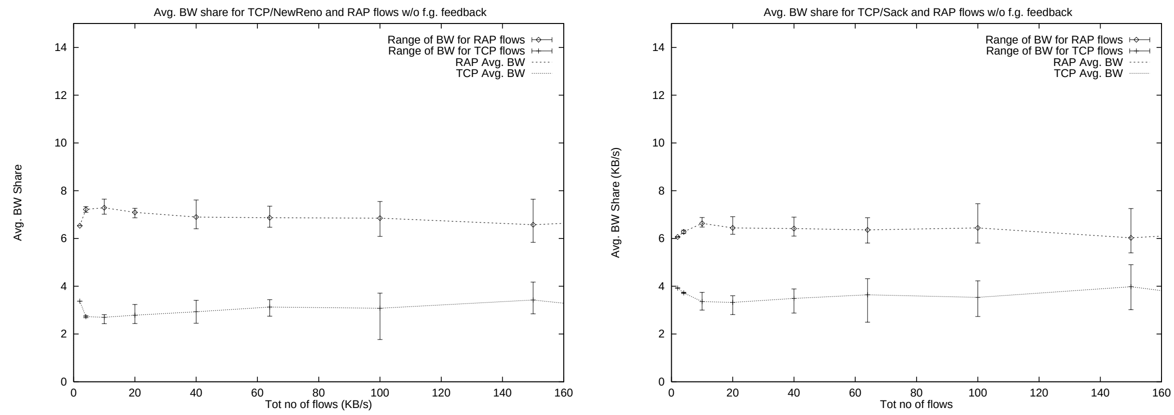
(b) RAP coexisting with Reno

Figure 5: Comparison of RAP with TCP(Tahoe and Reno)

shows the average bandwidth share of n RAP and n TCP Tahoe flows coexisting over the topology depicted in figure 4. The resources (i.e. the bottleneck bandwidth and the buffer size) are scaled up linearly with the total number of flows. The range of the bandwidth share among RAP and TCP flows are represented by vertical bars around the average value. This result implies that RAP is not terribly TCP-friendly across all the simulations. The observed unfairness against TCP traffic can be due to one of the following reasons: 1. TCP's inherent performance limitations, 2. An artifact of configuration parameters and 3. Unfairness imposed by coexisting RAP flows.

TCP suffers from some performance limitations[FF96, Mor97]. In particular, when TCP experiences multiple losses within a window or the window is smaller than 4, it is constrained to either wait for retransmission timeout or go through slow-start. As a result, TCP may temporarily lose its ack-clocking and its congestion control mechanism diverges from the AIMD algorithm. The severity of the problem varies among different flavors of TCP and mainly depends on window size and loss pattern. TCP Sack is able to recover from the multiple loss scenarios easier than other flavors of TCP whereas Reno's performance is substantially degraded [FF96]. Generally, TCP's ability to efficiently recover from multiple loss increases with its window size. The more TCP diverges from the AIMD algorithm, the less bandwidth it obtains and the lower performance it exhibits in large scale.

We exploited the difference among various TCP flavors to assess the impact of TCP's performance problem on the observed unfairness. We have repeated the same experiment with RAP against Reno, NewReno[11] and Sack TCP. Results are shown in Figure 5(b), 6(a), 6(b) respectively. These results confirm that the large-scale behavior of TCP traffic is in agreement with the behavior reported in[FF96] . These experiments also reveal that TCP's inherent performance problems partially contribute to unfairness to the TCP traffic. It is interesting to notice that the inter-protocol fairness remains unchanged across all simulations [12].



(a) RAP coexisting with NewReno

(b) RAP coexisting with Sack

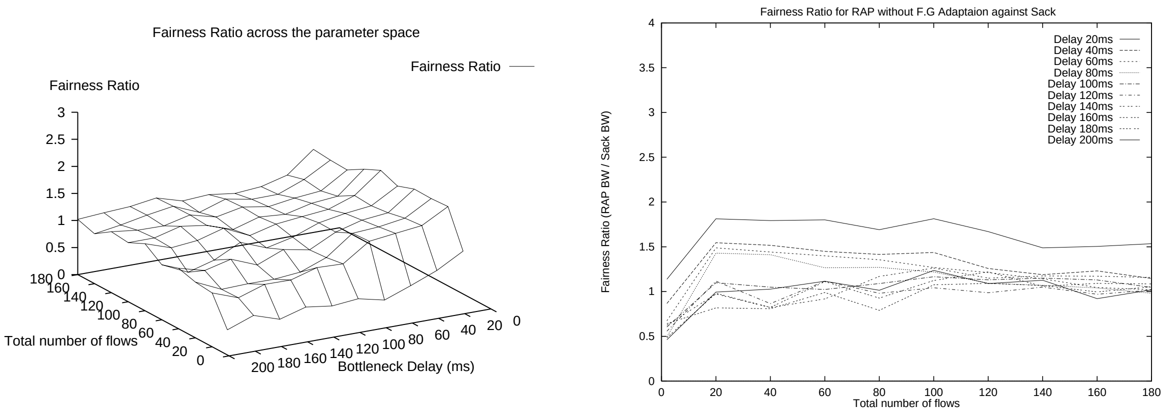
Figure 6: Comparison of RAP with TCP(NewReno and Sack)

We would like to limit the impact of the TCP's performance problem and focus on the interaction between RAP and TCP traffic. Therefore, we choose Sack as an ideal representative for TCP flows. For the rest of this paper whenever we refer to TCP, we mean Sack TCP otherwise it is explicitly stated. Since we are unable to exhaustively examine the parameter space, we focus our attention to those parameters that play key roles in protocols' behavior. RTT and TCP's congestion window are particularly important. RTT is a crucial parameter because it affects rate-adjustment in both RAP and TCP. TCP's congestion window is a primary factor in the performance of the TCP protocol. We introduce the term *inter-protocol fairness ratio* that is the ratio of the average RAP bandwidth calculated across all the RAP flows over the average TCP bandwidth calculated across all the TCP flows. We changed the delay of the bottleneck link to control the value of RTT, and the buffering was adjusted accordingly.

[11] NewReno is a modified version of Reno TCP that avoids some of the Reno's performance problems. For more details, refer to [FF96]

[12] Notice that some of our results seem to have minor phase effect, e.g. for 100 flows in figure 5(b).

Figures 7(a) and 7(b) depict the fairness ratio as a function of the bottleneck link delay and the total number of flows. Figure 7(b) provides the side view (from the delay axis) of figure 7(a) for easier comparison. Each data point is obtained from an experiment where half of the flows are RAP and the other half are Sack TCP. The bottleneck resources (i.e. the buffer size and the bandwidth) are linearly scaled up with the total number of flows and all other parameters are the same as previous simulations.



(a) Fairness Ratio across the parameter space

(b) Side view of the previous graph

Figure 7: Exploring the inter-protocol fairness across the parameter space

Figure 7(a) reveals several interesting trends in the fairness ratio as we traverse the parameter space. Some of these trends are the following; 1. For a particular value of the bottleneck delay, increasing the number of flows improves the fairness ratio except for the smallest value of delay(i.e. delay = 20ms) in which the ratio never converges to one. This special case is in fact, the problem that we have observed in the previous section. These figures illustrate that except for small simulations, RAP exhibits TCP-friendly behavior. The different behavior in small simulations has to do with TCP' burstiness and loss pattern in these scenarios. We will address these problems shortly. 2. Excluding the simulations with small bottleneck delay (i.e. $Delay < 60$) as well as small simulations, the fairness ratio is mostly close to one and is not a function of the bottleneck delay (i.e. RTT). The problem with short bottleneck delay in small simulations has to do with the small size of TCP's congestion window. In these scenarios, TCP has a smaller congestion window and frequently experiences multiple loss due to the lack of statistical multiplexing. As the bottleneck delay increases, both the bottleneck pipe size and the buffer size increase. [13] This allows TCP flows to have a larger number of packets on-the-fly and easily recover from multiple losses.

We conducted another set of simulations to observe the primary effect of TCP's congestion window on the fair-

[13]Note that we keep the ratio of the buffer size to the pipe size fix for the bottleneck link across the simulations. Thus the maximum queuing delay increases with the bottleneck delay. This in turn, could further increase the average RTT depends on the behavior of the aggregate traffic.

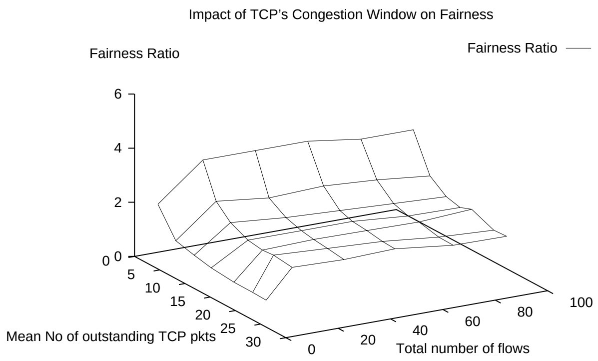


Figure 8: Variation of the Fairness ratio with TCP's congestion window

ness ratio. The congestion window is dependent on several parameters such as available bandwidth per flow, buffer size, mean queue size, queue management scheme and number of flows. We change the bottleneck bandwidth as a primary factor to control the value of congestion window. We decided to measure the number of outstanding TCP packets per flow instead of congestion window for two reasons. Firstly, TCP's congestion window may not be full during the fast-recovery period. In those cases, TCP's behavior depends on the number of outstanding packets. Secondly, since RAP is not a window-based mechanism, the number of packets on-the-fly seems to be the only common base of comparison from the network's point of view.

Figure 8 shows the variation of the fairness ratio as a function of the number of flows and the amount of allocated bandwidth per flow. Since the number of outstanding packets is dependent on both variables, we have used the mean number of outstanding packets (averaged across all the TCP flows in a simulation) as the x coordinate for the corresponding data point instead of the amount of allocated bandwidth per flow[14]. This graph clearly confirms our hypothesis that TCP's performance is directly influenced by the number of outstanding packets in transit. It shows that as the number of outstanding packet grows, the fairness ratio enhances except for simulations with a small number of flows (n=1). Therefore, under a heavy load, if the number of outstanding packets for a TCP flow drops below a threshold, its performance is degraded substantially. Under these circumstances, RAP can easily utilize the available bandwidth because it decouples congestion control from error control and only perform the former.

Figure 8 also implies that the number of coexisting flows does not have a visible impact on TCP's performance when resources scale appropriately, except for very small numbers of flows.

After exploring the parameter space in several directions, our evidence implies that "the RAP protocol behaves in a TCP-friendly fashion over a large portion of the space". The observed unfairness against TCP is mainly due to TCP's inability to recover from multiple losses with small congestion window. In other words, as long as TCP's congestion control does not diverge from the AIMD algorithm, TCP and RAP traffic evenly share the bandwidth.

[14]This slightly changes alignment of the graph.

The critical case is mainly observed when 1. load is high and TCP has only a small number of outstanding packets or 2. the number of flows is small and each one experience burst of loss due to lack of sufficient statistical multiplexing. While TCP is forced to slow down and perform error recovery, RAP simply abides by AIMD rules and is not sensitive to number of losses per RTT due to its cluster loss mechanism. Therefore, RAP obtains a larger share of the bandwidth.

For the rest of this section, first we examine the impact of the proposed fine-grain rate adaptation on TCP-friendliness. Then we revisit the problem that we frequently observed with small simulations.

4.2.2 Fine-grain rate adaptation

As we theorized that one reason TCP performed poorly was that ACK-clocking provides a degree of congestion *avoidance* which RAP did not duplicate, we investigated the effect of fine-grain rate adaptation in RAP.

We simulated fine-grain rate adaptation with two granularities; adapting once per-step and adapting once per-ack based on the RTT signal. Both performed acceptably, but per-ack adaptation is slightly better, and so those results are presented here.

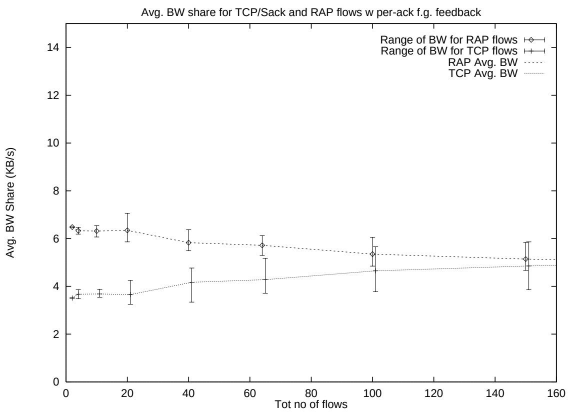
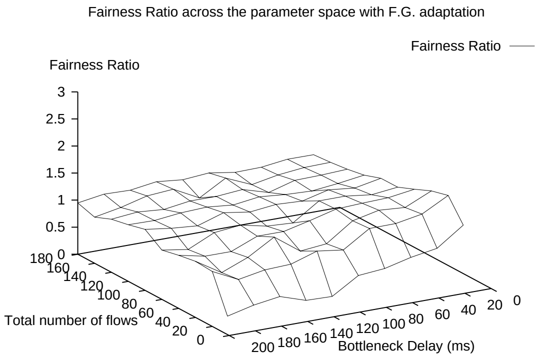


Figure 9: RAP with per-ack fine grain rate adaptation against Sack

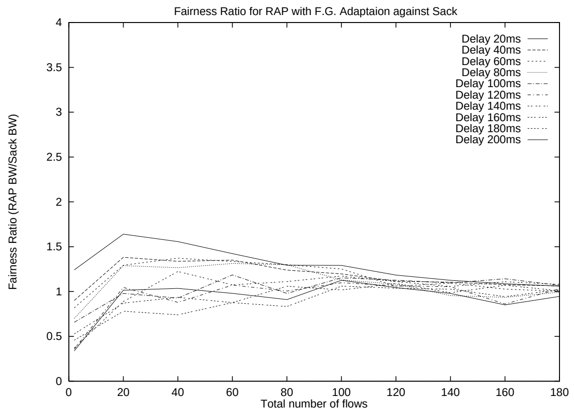
Figure 9 shows the average bottleneck bandwidth share for n Sack flows against n RAP flows with per-step fine-grain rate adaptation and 20ms bottleneck link delay. This demonstrates consistent improvement in RAP's behavior. Although the impact is marginal in small simulations, it becomes more visible as the simulation size grows. We have conducted the same experiment with Tahoe, Reno and NewReno. Our results show that fine-grain rate-adaptation improves the fairness particularly in large simulations.

It is interesting to note that fine-grain rate-adaptation also enhances the intra-protocol fairness among the RAP flows, i.e. vertical bars become smaller. This is probably because there are some vestiges of phase effect, and fine-grain feedback helps to eliminate these.

To increase our confidence in these results and evaluate the impact of the fine-grain rate-adaptation mechanism over a wide range of parameters, we explored the parameter space further. Figure 10(a) shows the fairness ratio as



(a) Fairness ratio across the parameter space



(b) Side view of the previous graph

Figure 10: Exploring the impact of fine-grain rate adaptation on fairness ratio

a function of the bottleneck link delay and the total number of coexisting flows where half of the traffic consists of RAP flows. Comparison of figure 10(a) with figure 7(a) reveals that the fine grain rate-adaptation only improves the fairness among connections with small RTT while it does not affect other areas. Figure 10(b) provides the side view of figure 10(a) for easier comparison.

This result implies that as long as TCP flows do not diverge from the AIMD algorithm, the fairness ratio is primarily determined by TCP's behavior and the large-scale behavior remains intact. This is indeed a desired property. However, for those scenarios where TCP traffic is vulnerable to multiple losses and achieves a smaller share of the bandwidth, the fine grain rate adaptation enhances resolution of rate adaptation for RAP flows by preventing them from overshooting the available bandwidth share. This in turn, reduces the probability of experiencing multiple losses across all flows. Consequently, TCP traffic obtains a fairer share of bandwidth.

4.2.3 Burstiness

We have observed two special cases where inter-protocol fairness was not achieved that have not been addressed yet. These cases are discussed in this section separately.

The first special case occurs in simulations with a relatively small number of (i.e. 10 to 50) flows over a bottleneck link with a small delay value(Figure 10(a)). Although this scenario is not usually exercised over the Internet the problem still deserves attention since RAP might be deployed over an ISDN line where it coexists with a small number of TCP flows.

The chief reason for the unfairness in these scenarios is TCP's burstiness. Since TCP's behavior is more bursty than RAP, TCP has a greater probability of experiencing loss due to buffer overflow. These losses tend to be bursty in small simulations because of the DropTail queues and lack of sufficient statistical multiplexing. As we mentioned earlier, TCP's congestion control algorithm can diverge from AIMD as a result of a bursty loss, while

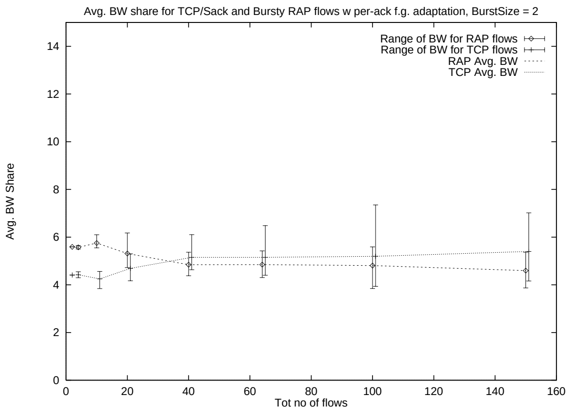
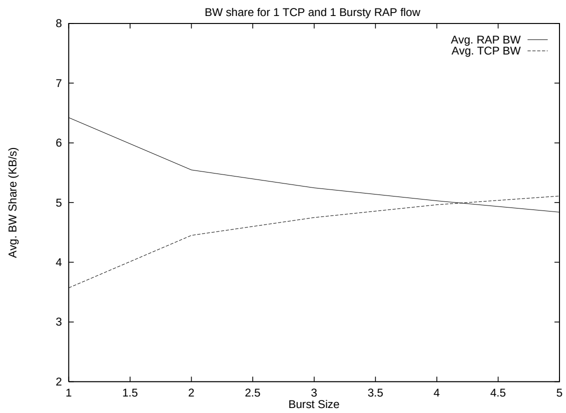


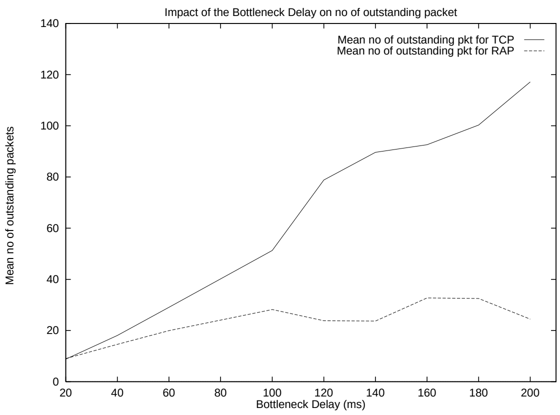
Figure 11: Bursty RAP with per-ack fine grain adaptation against Sack

RAP reacts to different loss patterns similarly because of the cluster loss mode. Thus on long time-scales, RAP obtains more bandwidth than TCP.

To observe the effect of burstiness, we changed RAP to send a burst of size b packets every $b * IPG'$ seconds. Figure 11 shows the average bandwidth share of n TCP Sack coexisting with n bursty RAP flows and 20ms bottleneck delay. The RAP flows perform fine-grain rate adaptation and the burst size is 2 packets. The resources are linearly scaled up as the number of flows increases. This graph demonstrates two points. Firstly, adding burstiness to RAP's behavior increases the probability of loss and experiencing back off. Secondly, as the simulation size grows and the level of statistical multiplexing increases, TCP's burstiness gradually disappears and TCP's performance improves. However, RAP's burstiness does not depend on the observed level of statistical multiplexing, and so as we move toward bigger simulations, TCP gradually outperforms this bursty RAP.



(a) Average bandwidth share for bursty RAP against TCP



(b) Mean number of outstanding packets for one RAP and one TCP

Figure 12: Impact of burstiness on RAP's behavior against TCP

We have repeated the previous simulation for one TCP flow against one bursty RAP against one for different burst sizes to assess the impact of burst size on performance. Figure 12(a) illustrates the average RAP and TCP bandwidth for these scenarios as the burst size increases. This figure clearly illustrates that burstiness is harmful to performance.

The second special case is observed when the fairness ratio among a very small number of flows monotonically decreases as the bottleneck delay increases. This phenomenon was observed in figure 10(a) and 7(a) in simulation with two flows(n = 1) as the bottleneck delay changes. We do not have enough evidence to provide a solid explanation for this case. However, we speculate that this has to do with impact of increasing buffer size on TCP's behavior. In the context of small simulations, if the buffer size at the bottleneck increases, TCP's burst can be absorbed at the bottleneck. This leads to a higher share of bandwidth for TCP.

Figure 12(b) depicts the number of outstanding packets for RAP and TCP flows in the same set of simulations. Notice that the buffer size linearly increases with the bottleneck delay [15]. This figure reveals that the number of outstanding packets for TCP rapidly increases with buffer size while it remains unchanged for RAP. As the buffer size increases, TCP manages to rapidly inflate its window and obtain a bigger share of the buffer. RAP is not as sensitive to buffer size as TCP because of its smooth transmission. Thus it simply operates with the left over portion of the bottleneck buffer. This special case requires further investigation.

4.2.4 RED Gateways

The main challenge here was to configure the RED gateway so that it behaves uniformly across all simulations. As we mentioned earlier, RED's performance closely depends on behavior of the aggregate traffic. Since this behavior could change with the number of flows, it is hard to obtain the same performance over a wide range without reconfiguring the gateway. Table 2 summarizes our configuration parameters :

Bottleneck BW	5KByte/s * No. of flows
Bottleneck Delay	20 ms
Buffer Size	12 * RTT * bottleneck b/w
Min. Threshold	5 packets
Max. Threshold	0.5 * Buffer
q_weight	0.002
max_p	Variable

Table 2

Half of the traffic consists of RAP flows with fine grain adaptation. We have provided sufficient buffer at the bottleneck point to eliminate any buffer overflow.

[15]Because we scale up the buffer size proportionally. It remains four times of the bottleneck pipe across our simulations

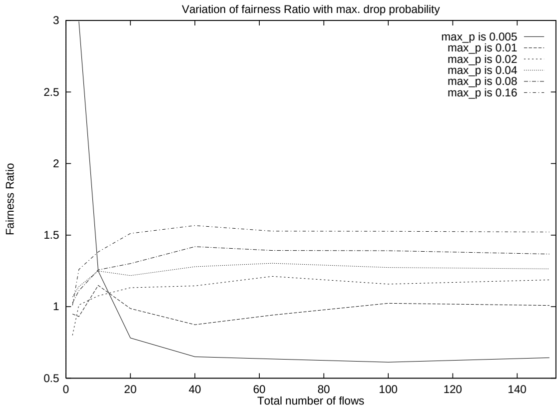


Figure 13: Impact of RED on the fairness

Figure 13 shows the fairness ratio for different value of max_p (maximum probability of loss) as the numbers of flows changes. This graph clearly illustrates three interesting points:

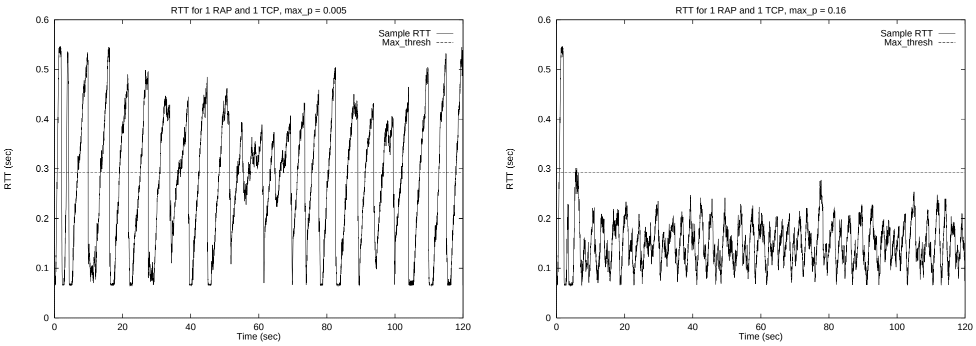
1. The behavior of the aggregate traffic is substantially different in small simulations.

2. Except for small simulations, the fairness ratio does not change with simulation size.

3. There exists a range for max_p where RAP and TCP evenly share the bottleneck bandwidth.

Figure 13 demonstrates that RED is able to evenly distribute the losses across the flows and avoid buffer overflow over a wide range. Thus RED has eliminated the unfairness caused by TCPs burstiness.

The higher the value of max_p, the more likely RED is to drop a packet when the buffer is less full, and so the lower the mean buffer utilization is. Figure 8 has already shown that TCP performs poorly with small values of *cwnd*, and higher values of max_p tend to reduce TCP's mean *cwnd*. RAP takes advantage of this, and a degree of unfairness results.



(a) RTT for 1 RAP, 1 TCP and max_p = 0.005 (b) RTT for 1 RAP, 1 TCP and max_p = 0.16

Figure 14: Impact of max_p on variation of queue length

As long as the average queue size remains in RED's operating region (below max_{th}), the bandwidth share between RED and TCP is quite fair. However, if the value of max_p is too small, the average queue size reaches max_{th}, and RED then starts dropping all packets until the average queue size decreases below max_{th} again. This process repeats and oscillations occur, with the loss probability alternating between max_p and 1.0. RED should not be operated in this region, and the curve in figure 13 shows this effect when $max_p = 0.005$. The differences between RAP and TCP are due to TCP's burstiness interacting with periodic oscillations of the average queue size about max_{th}. With small simulations, the oscillation period is long, and both TCP and RAP lose whole RTT worth of packets, but TCP takes a very long time to recover, while RAP recovers comparatively easily. With large simulations, the period of these oscillations is much shorter, and although a few TCP's may lose, on average a TCP is less likely to be hit by one of the loss periods than a RAP flow which spaces its packets out evenly. Hence, on average TCP does better than RAP. It should be emphasized that this RED regime will impose terrible loss bursts on real-time flows, and should be avoided at all costs. Figures 14(a), 14(b) graph the measured RTT for small simulations, and demonstrate these oscillations in figure 14(a) with $max_p = 0.005$ versus normal RED behavior in figure 14(b) with $max_p = 0.02$.

We conclude that with appropriate tuning, RED can help significantly with the fairness between RAP and TCP, but that aggressively pushing for very low buffer utilization is counter-productive when RAP and TCP share a link because TCP then diverges from AIMD.

4.2.5 Intra-protocol Behavior

Figure 15 shows the average bandwidth share among n RAP flows as we increase the number of flows while the amount of resources remain fixed. The bottleneck bandwidth is 250 KByte/s and the bottleneck buffer size is four times the bandwidth-bottleneck RTT product. The rest of the parameters are similar to table 1.

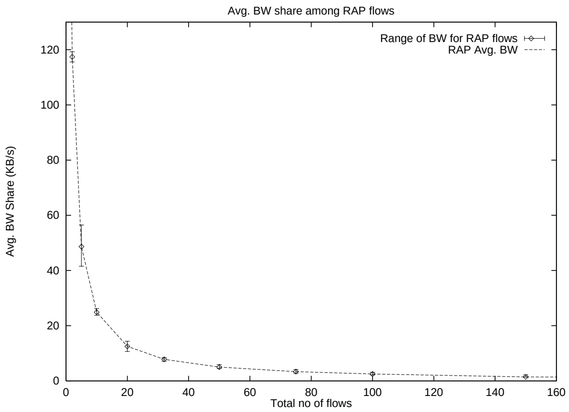
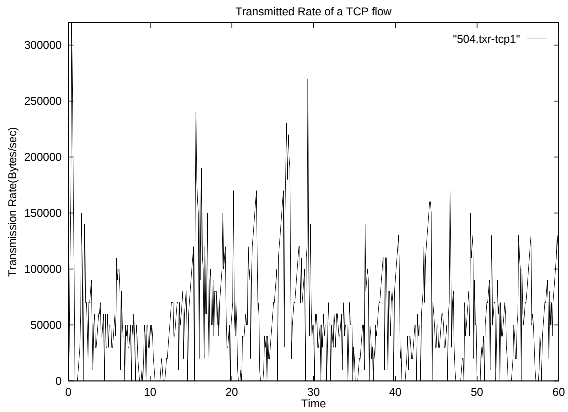


Figure 15: Intra-protocol fairness among RAP flows

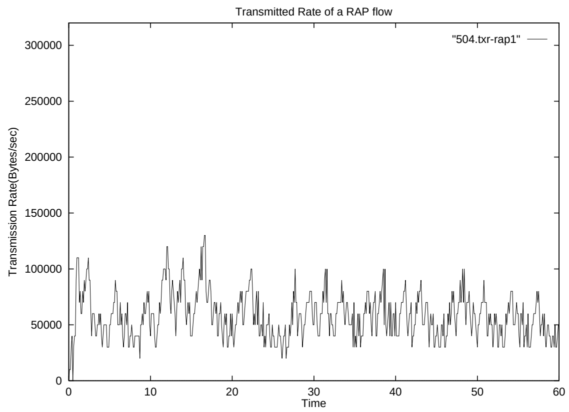
We performed a number of such simulations to show that RAP flows divide the bandwidth fairly for a wide range of network loads. Note that we have already covered the impact of load on inter-protocol fairness in Figure 8.

4.2.6 Smoothness of RAP transmission rate

Figures 16(b) and 16(a) show the variation of goodput for a sample RAP and TCP Tahoe[16] flow respectively from a simulation with 32 RAP and 32 TCP. Although the transmission rate of TCP flows has higher variance than that of RAP flows, both have the similar mean values.



(a) Transmission rate of a sample TCP flow



(b) Transmission rate of a sample RAP flow

Figure 16: Comparing the smoothness of transmission rate between RAP and TCP

Thus RAP satisfies our design goal of providing a smoother, more predictable congestion-control signal to our real-time applications than TCP does.

5 Conclusion and Future Work

In this paper we have presented a rate adaptation protocol and extensively examined its interaction with TCP through simulation. Although achieving TCP-friendliness over a wide range of network parameters is extremely challenging, RAP reasonably fulfill this goal. We have devised and evaluated a fine grain rate adaptation mechanism to emulate TCP's ack-clocking property. Our results show that the fine grain adaptation extends inter-protocol fairness to a wider range. Divergence of TCP's congestion control from the AIMD algorithm is often the main cause for the imposed unfairness against TCP in special cases. This problem is pronounced more clearly with Reno and Tahoe while it has a more limited impact on Sack. Toward this end, we observed that the bigger TCP's congestion window is, the closer it follows the AIMD algorithm. RED gateways with a proper configuration, can result in an ideal inter-protocol sharing.

We plan to continue our work in several directions. First, we have developed a prototype to examine RAP's performance in a real network. Two-way congestion scenarios is another interesting area for us to explore. We

[16]TCP Sack is smoother than Tahoe, so this illustrates near worst case behavior by TCP

have also developed a framework to apply RAP to layered-coded streams. This framework can be deployed as a source-based fine-grain layer adjustment policy for delivery of realtime stream. This approach is able to cope with bandwidth heterogeneity among receivers without recoding the stream for different receivers.

RAP is just a core component for an end-to-end transport protocol. We plan to integrate RAP with a compatible buffer management and play back algorithm and develop a complete end-to-end transport protocol for playback realtime streams. Among other interesting issues, we would like to explore different repair approaches such as retransmission and redundant coding. Moreover, we plan to investigate special requirements of interactive application and their implications on different components, e.g. RAP and buffer management, etc.

References

[Bol93] J. C. Bolot. Characterizing end-to-end packet delay and loss in the internet. *Journal of High Speed Networks*, 2(3):289–298, September 1993.

[CJ89] D. Chiu and R. Jain. Analysis of the increase and decrease algorithm for congestion avoidance in computer networks. *Journal of Computer Networks and ISDN*, 17(1):1–14, June 1989.

[CPW98] S. Cen, C. Pu, and J. Walpole. Flow and congestion control for internet streaming applications. *Proceedings Multimedia Computing and Networking*, January 1998.

[CTCL95] Z. Chen, S-M Tan, R. H. Campbell, and Y. Li. Real time video and audio in the world wide web. *Fourth International World Wide Web Conference*, December 1995.

[FF96] K. Fall and S. Floyd. Simulation-based comparison of tahoe, reno and sack tcp. *Computer Communication Review*, 26(3):5–21, July 1996.

[FJ92] S. Floyd and V. Jacobson. On traffic phase effect in packet-switched gateways. *Internetworking: Research and Experiences*, 3(3):115–156, September 1992.

[FJ93] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[Flo91] S. Floyd. Connections with multiple congested gateways in packet-switched networks. *Computer Communication Review*, 21(5):30–47, October 1991.

[Inca] Microsoft Inc. Netshow service, streaming media for business. http://www.microsoft.com/NTServer/Basics/NetShowServices.

[Incb] Vxtreme Inc. Vxtreme streaming video player. http://www.vxtreme.com.

[Jac88] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM*, pages 314–329. ACM, August 1988.

[Jac97] V. Jacobson. Email on the end-to-end email list. February 1997.

[Jai89] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM Computer Communication Review*, 19(5):56–71, October 1989.

[JE97] S. Jacobs and A. Eleftheriadis. Real-time dynamic rate shaping and control for internet video applications. *Workshop on Multimedia Signal Processing*, pages 23–25, June 1997.

[MF95] S. McCanne and S. Floyd. Ns (network simulator). 1995. http://www-mash.cs.berkeley.edu/ns.

[MF97] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control. *Technical note sent to the end2end-interest mailing list*, January 1997. http://www.psc.edu/networking/papers/tcp-friendly.html.

[MK92] P. Mishra and H. Kanakia. A hop by hop rate-based congestion control scheme. *ACM SIGCOMM*, 1992.

[MMFR96] M . Mathis, S. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. *RFC 2018*, April 1996.

[Mor97] R. Morris. TCP behavior with many flows. *IEEE Int'l Conf. on Network Protocols*, October 1997.

[MSMO97] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review*, 27(3), July 1997.

[Net] Progressive Networks. Http versus realaudio client-server streaming. http://www.realaudio.com/help/content/http-vs-ra.html.

[OK95] A. Ortega and M. Khansari. Rate control for video coding over variable bit rate channels with applications to wireless transmission. *Proceedings of the 2nd IEEE International Conference on Image Processing (ICIP)*, October 1995.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. *ACM SIGCOMM*, September 1998.

[She90] S. Shenker. A theoretical analysis of feedback flow control. In *ACM SIGCOMM*, pages 156–165, September 1990.

[TZ98] W. Tan and A. Zakhor. Error resilient packet video for the internet. *Submitted to Proceedings of the IEEE International Conference on Image Processing (ICIP)*, May 1998.