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*

Abstract—**End-to-end congestion control mechanisms have been critical to the robustness and stability of the Internet. Most of today's Internet traffic is TCP, and we expect this to remain so 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 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 an ideal fairness between TCP and RAP traffic.

I. Introduction

The Internet has recently been experiencing an explosive growth in the use of audio and video streaming. Such 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 is neither controllable nor predictable. Lack of support for QoS has not prevented rapid growth of realtime streaming applications and this is expected to continue. Many of these applications playback stored video or audio for a client over the network. Examples include continuous media servers, digital libraries, distant learning and shopping. These playback clients can afford to slightly delay the playback point and buffer some data to partially absorb variation of the network bandwidth and end-to-end delay.

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

In a shared network such as the Internet, all end-systems are expected to react to congestion by adapting their transmission rates, to avoid congestion collapse and to keep network utilization high[6]. 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. 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, it is crucial that realtime streams perform TCP-friendly congestion control. By this, we mean that a realtime flow should obtain approximately the same average bandwidth over the timescale of a session as a TCP flow along the same path under the same conditions of delay and packet loss [13].

We have been working on an architecture (fig. 1) for delivery of layered-encoded stored realtime streams over the Internet[20]. Our goal is to make realtime playback applications be good network citizens. A typical target application could be a web-server or a video-on-demand server that provides access to a variety of multimedia streams for a large number of heterogeneous clients. The idea is to separate congestion control from error (and 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 the Rate Adaptation Protocol (RAP) in a TCP-friendly fashion. The RAP module is exclusively in charge of congestion control and loss detection. The layer manager 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 layered encoding. The layer manager tries to deliver the maximum number of layers that can fit in the available bandwidth. Rate adaptation happens on a timescale of round-trip times but layers are added and dropped on a longer 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 de-

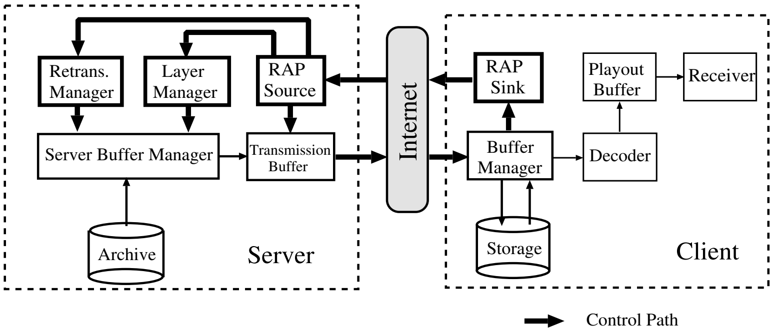

Fig. 1. RAP in a typical end-to-end architecture for realtime playback applications in the Internet

termined by the retransmission manager. Note that the aggregate bandwidth used by the server, including retransmission, should not exceed the bandwidth that is specified by RAP. This approach copes with bandwidth heterogeneity among clients without recoding the stream for each client.

Currently, most Internet realtime applications lack end-to-end congestion control or are not 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 solution would be to make realtime flows use reservations 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 at low cost. Even in a network that supports reservation, different users that fall into the same class of service or share a reservation still interact as in best effort networks. Thus we believe that congestion control for these applications is critical for the health of the Internet.

This paper presents the 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 playback of realtime streams as well as other semi-reliable Internet applications. 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 [4]. RAP adopts an AIMD algorithm for rate adaptation to achieve inter-protocol fairness and TCP-friendliness. RAP performs loss-based rate control and does not rely on any explicit congestion signal from the network since packet loss seems to be the only feasible implicit feedback signal in the Internet due to the presence of competing TCP traffic. However, if the network supported explicit congestion signaling, RAP could exploit this to behave more efficiently.

We extensively evaluated performance of RAP through simulation. Our results 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 this unfairness. 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. Our results show that deploying RED [7] 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 II. In section III, we present various aspects of the RAP protocol. Detailed description of our simulation results are presented in section IV. Finally, section V concludes the paper and discusses some of our future work.

II. Related Work

Congestion control is not a new topic and a large body of work has accumulated describing various mechanisms. However, the critical work for TCP-friendly congestion control mechanism in best-effort networks is somewhat more limited.

Jacob et al.[10] 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. Moreover, this scheme still inherits TCP's bursty behavior.

A common approach for rate adaptation is adaptive encoding through the adjustment of codec quantization parameters based on state of the network. Many of these studies have not addressed inter-protocol fairness; instead they strive to improve the perceptual quality. However, work in [23] proposes an adaptive coding scheme, using the formula presented in [13] and [15] that captures the macroscopic behavior of TCP. This shows promise, but it has yet to be shown that this formula or the more detailed variant of it in [18] can be used in a wide range of situations without introducing possible large-scale oscillatory behavior. Moreover, it is CPU-intensive for a server to adaptively encode a large number of streams simultaneously for all active clients.

Cen et al.[2] present the SCP protocol for media streaming. This is a 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 its rate adjustment mechanism using the shortest RTT that has been measured since this may widely vary for different flows.

There are many commercial media streaming players that are currently deployed over the Internet such as Realplayer[17] and Microsoft Netshow[9]. Although they claim to be adaptive, no analysis is available to verify any claims. Work in [3] describes the VDP protocol that is deployed in Vosaic. Their adaptation algorithm is clearly not TCP-friendly.

Our study differs from previous studies of realtime streaming over best-effort networks. We develop a rate adaptation mechanism that will result in inter-protocol fairness and TCP-friendly behavior. The majority of previous work either does not address fairness, or has not examined sufficient cases to find the bounds where they cease to be fair(e.g. [22]).

III. 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 [12]. These are the *decision function* , the *increase/decrease algorithm*, and the *decision frequency*.

Decision Function

The rate adaptation scheme can 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.

RAP considers losses to be congestion signals, and uses timeouts, and gaps in the sequence space to detect loss.

Similar to TCP, RAP maintains an estimate of RTT, called $SRTT$, and calculates the timeout based on the Jacobson/Karel's algorithm. However, it detects the timeout losses differently because RAP is *not* ack-clocked. Unlike TCP, a RAP source may send several packets before receiving a new ACK to update the RTT estimate. Thus RAP couples the timer-based loss detection to the packet transmission. Before sending a new packet, the source checks for a potential timeout among the outstanding packets using the updated value of the $SRTT$ estimate.

The ACK-based loss detection mechanism in RAP is based on the same intuition as fast-recovery in TCP. If a RAP source receives an ACK that implies delivery of three packets after the missing one, the packet is considered lost. RAP requires a way to differentiate the loss of an ACK from the loss of the corresponding data packet. We have added redundancy to the ACK packets to specify the last hole in the delivered sequence space and provide robustness against single ACK losses. Note that the timeout mechanism is still required as a back up for critical scenarios such as a burst of loss.

Increase/decrease Algorithm

RAP uses an AIMD increase/decrease algorithm. In the absence of packet loss, the transmission rate is periodically increased in a step-like fashion. 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) [11]:

$$S_i = \frac{PacketSize}{IPG_i}, \qquad IPG_{i+1} = \frac{IPG_i * C}{IPG_i + C} \tag{1}$$

$$\alpha = S_{i+1} - S_i = \frac{PacketSize}{C} \tag{2}$$

where S_i and α denote transmission rate and *step height* respectively. C is a constant with the dimension of time. Upon detecting congestion, the transmission rate is decreased multiplicatively, by doubling the value of IPG:

$$S_{i+1} = \beta S_i, \qquad IPG_{i+1} = IPG_i/\beta, \qquad \beta = 0.5 \tag{3}$$

Decision Frequency

Decision frequency specifies how often to change the rate. The optimal adjustment frequency depends on the feedback delay. Feedback delay in ACK-based schemes is equal to one RTT. It is suggested that rate-based schemes adjust their rates not more than once per RTT [13]. Changing the rate too often results in oscillation whereas infrequent change leads to unresponsive behavior.

RAP adjusts the IPG once every $SRTT$ using (1). The time between two subsequent adjustment points is called a *step*. 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 source to observe the reaction of the network to the previous adjustment before making a new adjustment. If the value of IPG is updated once every $SRTT$ and we choose the value of C to be equal to $SRTT$, the number of packets sent during each step is increased by 1 every step. Since the length of each step is $SRTT$ and the height of each step is inversely dependent on $SRTT$, the slope of the transmission rate is inversely related to $SRTT^2$.

$$Slope = \frac{StepHeight}{StepLength} = \frac{\alpha}{SRTT} = \frac{PacketSize}{C*SRTT} \quad (4)$$

$$C = SRTT \Rightarrow Slope = \frac{PacketSize}{SRTT^2} \quad (5)$$

TCP's slope of linear increase is related to RTT in the same way in the steady state. Thus a RAP source can exploit RTT variations and adaptively adjust its 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.

RAP is "unfair" to flows with longer RTT in the same way that inter-TCP unfairness has frequently been reported[8]. RAP connections with shorter RTTs are more aggressive and achieve a larger share of the bottleneck bandwidth. In general, other measures of fairness can be only achieved by implementing the required machinery in the network[21]. As long as the unfairness problem is not resolved among TCP flows, being TCP-friendly implies accepting this unfairness. Due to lack of space, we have not discussed startup behavior of RAP, but as it is designed for relatively long-lived sessions, its startup behavior is not crucial.

A. Clustered Losses

In a shared best-effort network with a high level of statistical multiplexing, the observed loss pattern has a near random behavior[1] 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 at most once per RTT as long as they react properly and promptly[13].

To achieve this, RAP requires a mechanism to identify a cluster of losses that are potentially related to the same congestion event. Right after loss of packet $Seq_{FirstLoss}$ that results in a back-off, the outstanding packets in the pipe, called a *cluster*, have a sequence number, Seq, within the following range:

$$Seq_{LastSent} \geq Seq > Seq_{FirstLoss} \quad (6)$$

where $Seq_{LastSent}$ is the last packet that has been transmitted. 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.

B. Fine-Grain Rate Adaptation

The AIMD rate adaptation algorithm does not necessarily result in a TCP-friendly behavior when TCP's performance is degraded due to heavy load. 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 short-term exponential moving average of the RTT captures short-term trends in congestion. However, we require a *dimension-less, zero-mean* feedback signal to be independent of the connection parameters and has a wider applicability. The ratio of the short-term to the long-term exponential moving average of the RTT signal exhibits these desired properties. We have exploited the RTT signal and devised a *continuous* 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 respectively.

At each *tuning* point, the value of IPG_i is modulated by the fine-grain feedback signal and the resulting value, IPG'_i, is used for the transmission timer:

$$IPG'_i = IPG_i * Feedback_i \quad (7)$$

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

C. Random Early Detection Gateways

There seems to be general agreement in the community on deploying Random Early Detection (RED)[7] gateways to improve both fairness and performance of TCP traffic. RED queue management tries to keep the average queue size low and, by preventing the buffer from overflowing, it also accommodates bursts of packets. One of the main problems for TCP's congestion control is to recover from multiple losses within a window [5]. This

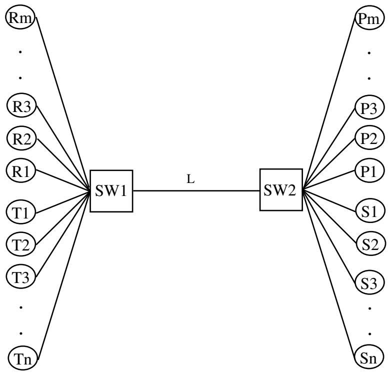

Fig. 2. Simulation Topology

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 retransmission timeout. Intuitively, as long as a RED gateway operates in its ideal region, RAP and TCP obtain an equal 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 and reacts only once to the first loss in an RTT.

We expect to observe substantial improvement in fairness by deploying RED even if it only prevents the buffer from overflowing and causing burst of loss. This behavior limits the divergence of TCP's congestion control from the AIMD algorithm.

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 traffic changes with time. Thus, configuration of RED is still a research issue.

IV. Simulation

In this section we present a summary of our simulation results. More details can be found in [19]. 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. Our simulations demonstrate that RAP is in general TCP-friendly. We have simulated RAP using the ns2 simulator [16], and compared it to TCP Tahoe, Reno, NewReno [5], Sack [14] and also run real-world experiments. Fig. 2 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. The switches implement FIFO scheduling and drop-tail queuing except in RED simulations. m RAP connections from sources $R_1...R_m$ 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 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 TCP flow control is not invoked. The average bandwidth for each flow is measured by the number of delivered packets during the last three quarters of the simulation time to ignore transient startup behavior. Simulation parameters are summarized in table 1.

Table 1

Packet Size	100 Byte
ACK Size	40 Byte
Bottleneck Delay	20 ms
B/W per Flow	5 KByte/s
B/W of Side Links	1.25 MByte/s
Tot. Delay of Side-Links	6 ms
Simulation Length	120 sec
TCP Maximum Window	1000
TCP Timeout Granularity	100 ms

A. 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 is a moving target. It's behavior changes drastically with configuration parameters and it has some internal constraints. During our simulations, with some exceptions, we attempted to minimize these problems by using 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 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 equally, the maximum queuing delay does not change across simulations. The impact of resource contention is also studied separately.

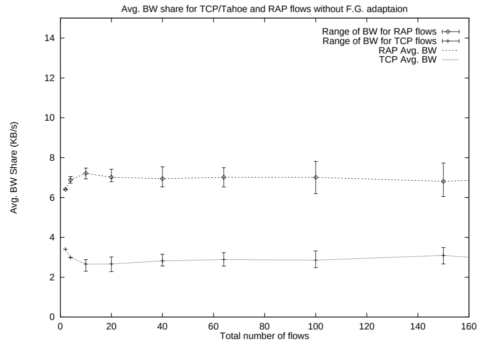

Fig. 3. RAP coexisting with Tahoe

3. We chose configuration parameters so that the TCP congestion window tends to be sufficiently large and TCP remains in its well-behaved mode.
4. We have explored a reasonable portion of the parameter space to examine inter-protocol fairness 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 those phenomenon that are purely related to TCP traffic.

B. Experiments and Results

B.1 TCP-friendliness

The first set of simulations examines the TCP-friendliness of RAP without fine-grain rate adaptation. Fig. 3 shows the average bandwidth share of n RAP and n TCP Tahoe flows coexisting over the topology depicted in fig. 2. 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 these simulations. The observed unfairness can be due to TCP's inherent performance limitations, an artifact of configuration parameters, or unfairness imposed by coexisting RAP flows.

TCP suffers from some performance limitations[5]. 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 patterns. TCP Sack is able to recover from the multiple loss scenarios easier than other flavors of TCP whereas Reno's performance is substantially degraded [5]. Generally, TCP's ability to efficiently recover from multiple losses increases with its window size. The more TCP diverges from the AIMD algorithm, the less bandwidth it obtains.

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[5] and Sack TCP. Our results confirm that the large-scale behavior of TCP traffic is in agreement with the behavior reported in [5]. These experiments also reveal that TCP's inherent performance problems partially contribute to unfairness. We would like to limit the impact of the TCP's performance problems and focus on the interaction between RAP and TCP traffic. Therefore, we chose TCP Sack as an ideal representative for TCP flows. For the rest of this paper whenever we refer to TCP, we mean TCP Sack unless explicitly stated otherwise.

Since we are unable to exhaustively examine the parameter space, we focus our attention on parameters that play key roles in protocols' behavior. RTT and TCP's congestion window are particularly important. RTT is crucial 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. The bandwidth was linearly scaled up with the total number of flows and the buffering was adjusted accordingly. Other parameters are the same as table 1. Fig. 4 depicts the fairness ratio as a function of the bottleneck link delay and the total number of flows. Each data point is obtained from an experiment where half of the flows are RAP and the other half are Sack TCP. This reveals several interesting trends in the fairness ratio:

For a particular value of the bottleneck delay, increasing the number of flows improves the fairness ratio except for the smallest value of delay (20ms) in which the ratio never converges to one. This figure illustrates that except for small simulations, RAP exhibits TCP-friendly behavior. The different behavior in small simulations has to do with TCP's burstiness and loss pattern in these scenarios[19].

Excluding simulations with a small bottleneck delay as well as small simulations, the fairness ratio is mostly close to one and is not a function of the RTT. The prob-

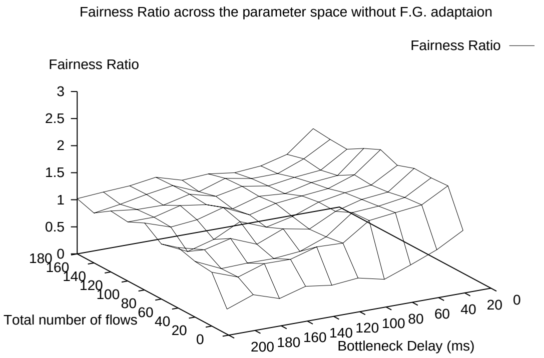

Fig. 4. Fairness Ratio across the parameter space

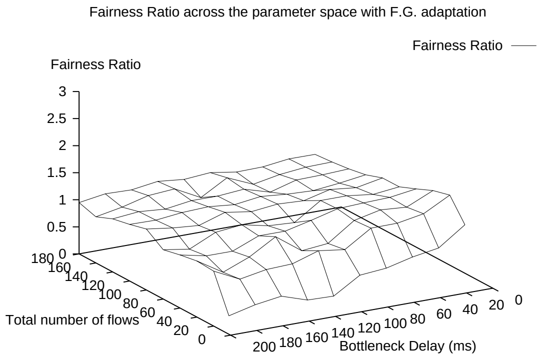

Fig. 6. Fairness Ratio across the parameter space

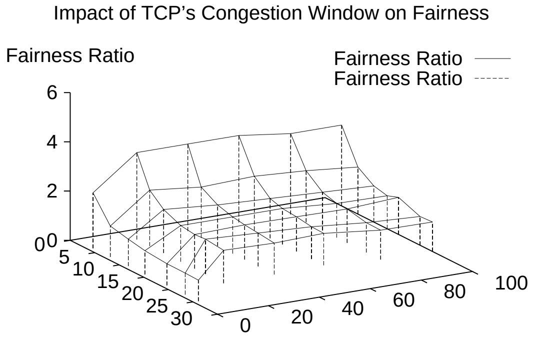

Fig. 5. Variation of the Fairness ratio with TCP's congestion window

lem 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 retransmission timeout. As the bottleneck delay increases, both the bottleneck pipe size and the buffer size increase. This allows TCP flows to have a larger number of packets on-the-fly and maintain their ack-clocking. We conducted another set of simulations to observe the primary effect of TCP's congestion window on the fairness 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 adjust 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. Fig. 5 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. This graph clearly confirms our hypothesis that TCP's performance is directly influenced by the number of outstanding packets in transit. As the number of outstanding packets grows, the fairness ratio improves 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 substantially degraded. Under these circumstances, RAP can easily utilize the available bandwidth because it decouples congestion control from error control and only performs the former.

Fig. 5 also implies that the number of coexisting flows does not have a visible impact on fairness when resources are scaled appropriately, except for very small numbers of flows.

B.2 Fine-grain rate adaptation

We have theorized that fine-grain rate adaptation attempts to emulate a degree of congestion *avoidance* that TCP obtains due to ack-clocking. To investigate the effect of fine-grain rate adaptation on TCP-friendliness, we explored the parameter space over a wide range. Fig. 6 shows the fairness ratio as a function of bottleneck link delay and the total number of coexisting flows. Half of the traffic consists of RAP flows. Comparison with fig. 4

reveals that fine-grain rate adaptation only improves the fairness among connections with small RTT (i.e. small TCP window) while it does not affect other areas. 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 loss of ack-clocking 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 loss of ack-clocking across all the TCP flows. Consequently, TCP traffic obtains a fair share of bandwidth.

B.3 RED Gateways

The main challenge here was to configure the RED gateway so that it behaves uniformly across all simulations. RED's performance closely depends on the 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:

Table 2

Min. Threshold	5 Packets
Max. Threshold	0.5 * Buffer
Bottleneck B/W	5 KByte/s * No. of Flows
Bottleneck Delay	20 ms
Buffer Size	12 * RTT * Bottleneck B/W
q_weight	0.002

Half of the traffic consists of RAP flows with fine-grain adaptation. We provided sufficient buffer at the bottleneck to eliminate buffer overflow. Fig. 7 shows the fairness ratio for different value of max_p (i.e. maximum probability of loss) as the number of flows changes. This graph clearly illustrates three interesting points:

1. There exists a range for max_p where RAP and TCP evenly share the bottleneck bandwidth.
2. Except for small simulations, the fairness ratio does not change with simulation size.
3. The behavior of the aggregate traffic is substantially different in small simulations.

Fig. 7 demonstrates that RED is able to evenly distribute the losses across all the flows and avoid buffer overflow over a wide range. Thus RED has eliminated the unfairness caused by TCP's burstiness. The higher the value of max_p, the more likely RED is to drop a packet before the buffer becomes full, and so the lower the mean buffer utilization is. Fig. 5 has already shown that TCP performs poorly with small congestion window, and higher

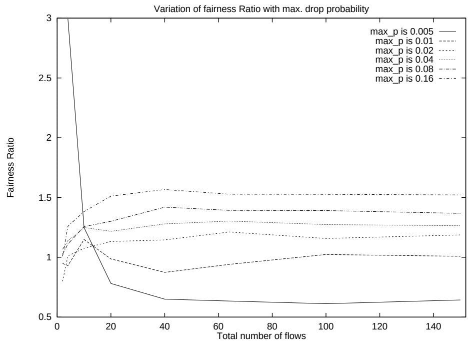

Fig. 7. Impact of RED on the fairness

values of max_p tend to reduce TCP's mean congestion window. RAP takes advantage of this, and a degree of unfairness results. 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 one. RED should not be operated in this region, and the curve in figure 7 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. 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 performs better than RAP. It should be emphasized that this RED regime will impose terrible loss bursts on realtime flows, and should be avoided at all costs. Figures 8 and 9 graph the measured RTT for small simulations, and demonstrate these oscillations in fig. 8 with $max_p = 0.005$ versus normal RED behavior in fig. 9 with $max_p = 0.16$. We conclude that, with appropriate tuning, RED can significantly improve the fairness between RAP and TCP. However that aggressively pushing for very low buffer utilization is counterproductive when RAP and TCP share a link because TCP then diverges from AIMD.

V. Conclusions and Future Work

We have presented a rate adaptation protocol and extensively examined its interaction with TCP through simula-

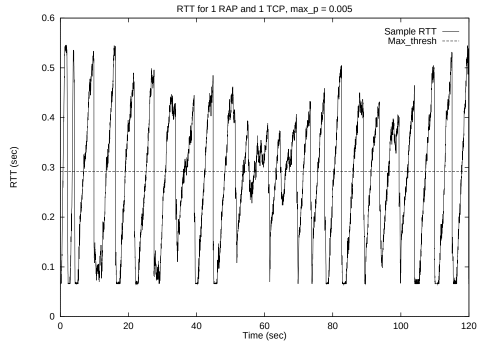

Fig. 8. RTT for 1 RAP, 1 TCP and max_p = 0.005

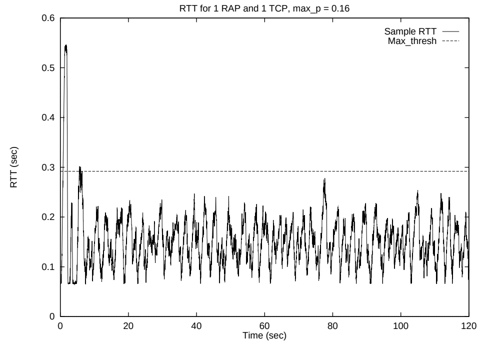

Fig. 9. RTT for 1 RAP, 1 TCP and max_p = 0.16

tion. Although achieving TCP-friendliness over a wide range of network parameters is extremely challenging, RAP reasonably achieves this goal. We devised and evaluated a fine-grain rate adaptation mechanism to emulate TCP's ack-clocking property. Our results show that the fine-grain rate 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 unfairness to TCP in special cases. This problem is pronounced more clearly with Reno and Tahoe while it has a more limited impact on Sack. We observed that the bigger TCP's congestion window is, the closer it follows the AIMD algorithm. Properly configured RED gateways can result in an ideal inter-protocol sharing.

We plan to continue our work in several directions. We have developed a prototype to examine RAP's performance in a real network. RAP is just a core component of the end-to-end architecture (fig. 1) for unicast playback of realtime streams over best-effort networks. We have also developed a "quality adaptation" mechanism that smoothly adjusts the quality of a layered encoded playback video while its transmission rate is controlled by RAP.

References

[1] 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.

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

[3] 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.

[4] 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.

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

[6] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *Unpublished*, February 1998. http://www-nrg.ee.lbl.gov/floyd/papers.html/end2end-paper.html.

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

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

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

[10] 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.

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

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

[13] 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.

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

[15] 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.

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

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

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

[19] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-end rate-based congestion control mechanism for realtime streams in the internet. *Technical report 98-681, CS-USC*, August 1998, http://netweb.usc.edu/reza/papers/rap.html.

[20] R. Rejaie, M. Handley, and D. Estrin. Architectural considerations for playback of quality adaptive video over the internet. *Technical report 98-686, CS-USC*, November 1998.

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

[22] D. Sisalem and H. Schulzrinne. The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme. *Workshop on Network and Operating System Support for Digital Audio and Video*, July 1998.

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