ELSEVIER

Computer Communications 25 (2002) 1230–1242

computer communications

www.elsevier.com/locate/comcom

Predictive flow control for TCP-friendly end-to-end real-time video on the Internet

Yeali S. Sun[a,*], Fu-Ming Tsou[b,1], Meng Chang Chen[c,2]

[a]*Department of Information Management, National Taiwan University, Taipei, Taiwan, ROC*
[b]*Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan, ROC*
[c]*Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC*

Received 21 December 2000; revised 1 November 2001; accepted 12 November 2001

Abstract

In order to cope with time-varying conditions in networks with no or limited QoS support like the current Internet, schemes have been proposed for real-time applications to dynamically adjust traffic sources' data sending rate. However, employing adaptive rate control may not be sufficient to prevent or handle network congestion. As most of the real-time applications are based on RTP/UDP protocols, an issue of possibly unfair sharing of bandwidth between TCP and UDP applications has been raised. In this paper, we propose an application-level control protocol called Real-time Rate and Retransmission Control Protocol Plus in which several control mechanisms are used and integrated to maximize the delivery performance of UDP-based real-time continuous media over the Internet while friendly sharing network bandwidth with TCP connections. Here we propose to use adaptive filters in network state characterization and inference. Both simulation and actual implementation performance results show that recursive least square-based adaptive prediction makes good use of past measurement in forecasting future condition and effectively avoids network congestion. It also shows that the scheme achieves reasonably friendly resource sharing with TCP connections.

Keywords: Flow/congestion control; End-to-end real-time video; Prediction; TCP-friendly

1. Introduction

Although there are many on-going research on end-to-end QoS guarantee for the next generation Internet [1–4], it will take some time to have such networks and services available in general. In the meantime, there are growing interests and activities on deploying multimedia services, including real-time audio/video clips on the WWW, electronic commerce, IP-telephone and Web TV over the existing Internet. How to maximize the delivery quality of these streaming applications in a best-effort network while 'friendly' sharing bandwidth with non-real time applications like TCP has become an important issue.

In this paper, we consider real-time flow and congestion control for stored multimedia applications, specifically MPEG video over the Internet. Works on real-time stored video transport in the past have focused on how to send variable-bit-rate video streams over a constant-bit-rate communication channel, e.g. Refs. [5–8]. They all assume resource reservation and QoS support are supported in the network. Recently, there were studies on stored variable-bit-rate video over best-effort networks [9–14]. To cope with time-varying network loads, these schemes focus on how a real-time traffic source dynamically adjust its transmission rate to prevent from sending excessive traffic into the network.

Such rate adjustment is all based on the feedback information from the receiver(s). They do not consider the effect of such adjustment on other types of traffic, e.g. TCP connections in the network. Differences between these schemes are mainly in two areas: (a) characterization of network state so the rate adjustment decision can be reliably made; (b) the effects of such adjustment on the flow itself and the efficiency of network bandwidth.

Employing adaptive rate control for real-time continuous media transmission may not be sufficient. Congestion control and avoidance have become an important issue in the Internet due to possibly unfair bandwidth sharing caused by the very different natures of the protocols used by real-time and non-real time traffic. In general, real-time applications are

* Corresponding author. Tel.: +886-2236-30231; fax: +886-2362-1327.
E-mail addresses: sunny@im.ntu.edu.tw (Y.S. Sun), fmtsou@eagle.ee.ntu.edu.tw (F.-M. Tsou), mcc@iis.sinica.edu.tw (M.C. Chen).
[1] Tel.: +886-2236-35251x554; fax: +886-2236-38247.
[2] Tel.: +886-2278-83799x1802; fax: +886-2278-24814.

0140-3664/02/$ - see front matter
PII: S0140-3664(02)00002-6

transported using RTP and UDP protocols, whereas most non-real time traffic is based on the TCP protocol. The former implements no congestion control, while the latter does. Research results have shown that this may lead to unfair sharing of bandwidth in the two types of traffic. In Ref. [15,16], the authors showed that TCP connections may be starved if most of the bandwidth is taken by UDP/RTP traffic. This is mainly due to the lack of congestion control in UDP and the low update frequency of the sender reports and receiver reports in RTP. On the other hand, when the network experiences congestion, a TCP sender will reduce its transmission window by half (in the Reno version) or even down to one packet (in the Tahoe version) in *one* round-trip time (RTT). Note that round-trip delays are typically smaller than the time interval between RTCP control packets, e.g. 5 seconds. If a UDP flow cannot react to the congestion as responsive as TCP flows, it may continue to send more data and capture an arbitrarily large fraction of the link bandwidth at the cost of other traffic. Therefore, it is essential that the UDP-based real-time transport protocol be aligned with TCP congestion control in the presence of network congestion.

A main challenge in the design of flow/congestion control scheme for UDP-based real-time applications is how to make UDP flows behave as good 'network' citizens—consuming only their fair share of bandwidth as with TCP traffic at the bottleneck link given that they have rigid delay and timing requirements. In this paper, we propose an application-level control protocol called Real-Time Rate and Retransmission Control Protocol Plus (R^3CP^+) in which several control mechanisms are used and integrated to maximize the delivery performance of UDP-based real-time continuous media over the Internet while these flows can friendly share network bandwidth with TCP connections. Note that when congestion occurs, all the flows affected will immediately experience performance degradation. Two key issues are raised here. The first issue is how to minimize such negative effect on real-time video sessions, especially when all the senders might be forced to reduce sending rates to relieve congestion. Second, can real-time sessions behave smarter to more effectively avoid congestion. Two methods are proposed here. The first method is to take the receiving buffer as a reserve bank. A video source will make use of the unused bandwidth available when the network is in an unloaded or lightly loaded condition to download additional packets to the receiver's buffer. Such packet store will help the receiver to cope with supply shortage during congestion. In this paper, a target minimal queue length of the receiving buffer is periodically recalculated according to the network state. The second method is to exercise selective transmission at the sender if the *requested sending rate* is less than the *desired sending rate* during congestion. In selective transmission mode, the sender only transmits packets with higher levels of significance, e.g. I frames. In the protocol, we also set the flow control period in the order of a RTT so to assure both UDP sessions and TCP connections act on congestion avoidance and control at the same time scale.

Another important issue in designing feedback-based congestion control mechanism is how to characterize network state and complement current state measurement so to achieve better flow control and avoid congestion. In R^3CP^+, we show the effectiveness of adaptive filters in network state forecase. Methods such as moving average and exponential average have been commonly used. In these schemes, the weighting factors of the current and past information are constant which limits the ability for systems to quickly adapt to network state changes while retaining network stability. Different from Ref. [8], we use an M-step adaptive linear predictor called recursive-least-square (RLS) predictor [17] to forecast network state. In RLS predictor, the weighting factor is corrected every time a new measurement was taken. Particularly, it uses the estimation error between current measurements and previous prediction of them to adjust itself. It not only can respond to network dynamics quickly but also remains stable.

The rest of the paper is organized as follows. In Section 2, we describe the recursive prediction algorithm in network state characterization and forecast. The least mean-square Kalman Filter is used to estimate packet loss probability and RTT. In Section 3, the algorithms that compute the desired sending rate and requested sending rate are presented. The flow and congestion control schemes for real-time video sessions are also described in detail. In Section 4, the performance of the scheme is evaluated via simulation, and the results are analyzed. The protocol was also implemented in a MPEG video player/browser running on Windows95. Some performance data run over the Internet are presented. Finally, Section 5 gives a conclusion.

2. Characterization and recursive prediction of network state

The characterization and inference of network state is performed by the receiver of a real-time video session and works as follows. Initially, before the sender starts sending the data, the receiver will *probe* the network in order to set proper initial values of several system parameters of the session, including the minimal amount of pre-downloaded data and the retransmission time interval for in-time packet recovery. Specifically, during the session setup phase, when the sender receives an acknowledgment of session establishment from the sender, it will send a number of *Network Probe packets* to collect the RTT and packet loss information between the sender and receiver. The RTTs are used to compute the *minimal* amount of data necessary to be pre-downloaded to the receiver's buffer before the playback starts. The goal is to make use of such preloading to accommodate volatile delay variations to ensure smooth playback during the session. During the data transfer phase, receive

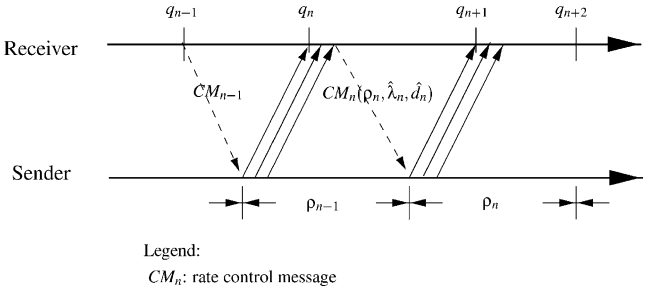


Fig. 1. Adaptive flow control between sender and receiver.

tries to maintain the data store in the receiving buffer at an adaptive target size. It continuously monitors packet receiving status and periodically sends *flow control packets* to the sender to instruct how fast or slow it should send the data. Sender always transmits packets at the rate as specified in the flow control packet.

In Ref. [11], we showed that integrating rate control with 'in-time' packet retransmission can significantly improve overall performance. On the contrary to the widespread belief that 'Retransmission of lost packets is unnecessary for continuous media applications as late packets are of no use as lost packets', we show that in best-effort networks, a major cause of performance degradation is due to packet loss, especially in the case of congestion. Therefore, as long as retransmitted packets can arrive at the destination before the deadline, the overall viewing quality can be greatly improved in particular for those packets carrying important information like GOP in MPEG 2 video stream. The RTT measured during each flow control period is used to adjust retransmission time interval.

Here, two parameters are used to describe the state of a transmission path: packet loss probability and RTT. The packet loss probability is obtained based on the packet receiving throughput and the expected amount of data inferred by the receiver. The RTTs are measured through flow control packets. Each parameter is recursively predicated by an *adaptive Kalman filter*. Based on the forecast and the desired sending rate, receiver computes the requested sending rate. The design rationale is to let the receiver decide based on its receiving and playout conditions, at what rate it would like the sender to send the data so it could share bandwidth with TCP connections in a 'reasonably friendly' way while maximizing its playback performance.

2.1. Prediction of packet loss probability

The end-to-end adaptive flow control is performed at discrete time instants. The time interval between two consecutive flow control points is called a *flow control period*. At the end of each flow control period, the receiver will send a *flow control message* to the sender specifying a new *rate* at which the sender should send the packets, i.e. the requested sending rate. Along with it are two other information: the desired sending rate and the estimate of the duration of the flow control period. If the requested sending rate is less than the desired sending rate, selective transmission is performed. Upon receiving a flow control message, the sender immediately adjusts its sending rate to the new requested rate. If network is stable, receiver expects packets sent under the new rate to start to arrive after one RTT (see Fig. 1). The duration of flow control periods are on the order of one RTT. This is to ensure the congestion control of UDP flows is at the same time scale as that in TCP.

The receiver computes packet loss probability at the end of each flow control period, which is defined as follows:

$$p_n = 1 - \frac{\gamma_n}{\rho_{n-1}}, \qquad n = 1, 2, \ldots, \tag{1}$$

where γ_n is the packet receiving rate measured during the nth control period and ρ_{n-1} is the requested sending rate sent at the $(n-1)$th period. For end-to-end adaptive flow control, if the decision is made solely based on the current state, a traffic source may over-react to transient changes of network loads. Rapid oscillation of rate adjustment can easily cause instability of the end systems as well as the network. This indeed has severe impact on the loss and delay jitter performance of real-time sessions. Consequently, it is important that the receive be able to capture both the *trend* and transience of the network load and instruct the sender accurately and cautiously. Special attention should be paid especially to the detection of congestion, avoiding overlook of congestion and increasing undue rate. These may result in bandwidth starvation of TCP connections.

Assuming measured packet loss probabilities form a random process. Here, a fixed amount of transmission history is taken into account in the estimation of packet loss probability. Let the size of the memory of the Kalman filter be fixed and denoted as M_{loss}. This is the amount of previously measured data recorded. Let $\underline{P}(n)$ be a vector random variable defined as follows:

$$\underline{P}(n) = [p_n, p_{n-1}, \ldots, p_{n-M_{\text{loss}}+1}]^T. \tag{2}$$

Next, we want to find a weighted vector $\underline{w}_p(n)$ such that we can predict $\hat{p}_{n+1}$ given the memory $\underline{P}(n-1)$ and the newly measured data p_n. The error between estimation and measurement is corrected by defining a forward prediction error parameter α_n, i.e.

$$\alpha_n = p_n - \hat{p}_n, \tag{3}$$

where $\hat{p}_n$ is our previous estimate. With a new measured data, the estimator is corrected as follows:

$$\underline{w}_p(n) = \underline{w}_p(n-1) + \underline{k}_p(n)\alpha_n, \tag{4}$$

where $\underline{k}_p(n)$ is the Kalman gain, defined as:

$$\underline{k}_p(n) = \frac{\theta^{-1}\mathbf{P}(n-1)\underline{p}(n)}{1 + \theta^{-1}\underline{p}^T(n)\mathbf{P}(n-1)\underline{p}(n)}, \tag{5}$$

and

$$\mathbf{P}(n-1) = \theta^{-1}\mathbf{P}(n-2) - \theta^{-1}\underline{k}_p(n-1)\underline{p}^T(n-1)\mathbf{P}(n-2), \tag{6}$$

where θ is the memory factor whose value is no greater than one but typically is very close to one. Typically, $M_{\text{loss}} \approx 1/(1-\theta)$. The estimate of $(n+1)$th is the minimum mean-square estimate of the state p_{n+1} from Eqs. (3) and (4). The estimate of the packet loss probability for the next flow control period is thus as follows:

$$\hat{p}_{n+1} = \underline{w}_p^T(n)\underline{p}(n). \tag{7}$$

2.2. Prediction of round-trip time for the flow control period

One of the main problems that causes TCP bandwidth starvation when sharing a bottleneck link with UDP flows is that when TCP connections reacts to congestion by throttling their congestion window, UDP flows consider the newly available bandwidth as an excess. In order to be 'TCP-friendly', it is important that the frequency of flow or congestion control of real-time UDP sessions be synchronized with that of TCP. That is, the flow control period should be in the order of one RTT so to assure both UDP and TCP connections act on congestion avoidance and control at the *same* time scale. The estimate of RTT is also used in the computation of the desired sending rate, and the number of packets to skip in selective transmission.

The measurement of RTT is performed as follows. A field is defined in the RTP header extension to distinguish between packets sent under different flow control periods. When a flow control message is sent, the receiver records its sending time. Upon receiving a flow control message from the receiver, the sender will immediately adjust its packet sending rate and change the group indication field in the RTP header extension to a new value to indicate the group of packets sent under the new rate. When the receiver receives the first packet of a new group, it timestamps the arrival time. The difference between this arrival time and the sending time of the flow control packet gives a new measurement of the RTT between the receiver and the sender.

The estimation of RTT $\hat{d}_n$ follows the same approach as in the prediction of packet loss probability [17]. The vector random variable $\underline{d}_n(n) = [d_n, d_{n-1}, \ldots, d_{n-M_{\text{dura}}+1}]^T$ is used to record the history of RTTs. Parameter $\beta_n = d_n - \hat{d}_n$ is used to correct the estimation error. The estimate of RTT is obtained as follows:

$$\hat{d}_{n+1} = \underline{\hat{w}}_d^T(n)\underline{d}(n), \tag{8}$$

where $\underline{\hat{w}}_d(n) = \underline{\hat{w}}_d(n-1) + \underline{k}_d(n)\beta_n$.

3. Adaptive flow control—take receiving buffer as a reserve bank

Note that when congestion occurs, all the flows affected would immediately experience performance degradation. Two key issues are raised in the design of congestion control scheme for real-time video streams. The first issue is that how to minimize such negative effect on the continuous playback performance of a video session given that during congestion fewer packet arrivals are expected. Second, can real-time flows behave smarter to more effectively avoid congestion. Two methods are proposed in this paper. The first method is to have the sender transmit or pre-store more data at the receiver side when the network is in an unloaded or lightly loaded condition. When the state of congestion is forecasted (i.e. such tendency has been inferred by the receiver), the sender will refrain its packet sending via *early* cease of rate increase to avoid congestion and taking a more drastic rate decrease to quickly relieve congestion. This is different from conventional methods used in TCP in which window size is continuously increased until congestion occurs and reduced afterwards. The idea is to make the receiving buffer a reserve bank. A video source will make use of the unused bandwidth available in the network to download additional packets to receiver's buffer whenever allowed. These additional packets will help the receiver to cope with supply shortage from the sender and maintain continuous and smooth playback. In this paper, a minimal amount of the target queue size is periodically recalculated according to the network state.

The second method is to exercise selective transmission at the sender. During a flow control period, if the requested sending rate is less than the desired sending rate, the sender will calculate the amount of data allowed to transmit. It only transmits packets with higher levels of significance, e.g. I frames. Our protocol adopts the principle of application level framing [18] in the fragmentation of MPEG video stream [19,20]. To play a video stream in real-time across the Internet, it is pre-parsed to generate a meta data file. The file describes the semantic data structures of the streams and contains information such as stream resolution, nominal frame rate, frame pattern, frame size and frame boundary. The meta data file is used in selective transmission and the segmentation and packetization of video frames at the sender. To pick up packets to skip we start from those of B type, then P type and so on. Among packets of the same type, frames are randomly chosen in such a way to avoid burst removal.

3.1. Receiver's desired sending rate and target queue length

In the following, we present the method that computes the desired sending rate. The idea is to take the receiving buffer as a reserve bank and loaded with *just enough* amount of data—neither too much to excessively increase buffer space requirement nor too few not to be able to cope with

Table 1
Determination of the requested sending rate

Current state/forecast	UNLOADED	LOADED	CONGESTED
UNLOADED	$\min(\hat{\lambda}_{\max}, \rho_{n-1} + \Delta_{\text{inc}})$	$\min(\hat{\lambda}_{\max}, \rho_{n-1} + \delta_{\text{inc}})$	$\min(\hat{\lambda}_n, \rho_{n-1} + \delta_{\text{inc}})$
LOADED	$\min(\hat{\lambda}_n, \rho_{n-1})$	$\min(\hat{\lambda}_n, \rho_{n-1})$	$\min(\hat{\lambda}_n, \rho_{n-1})$
CONGESTED	$\min(\hat{\lambda}_n, \rho_{n-1} \times \delta_{\text{dec}})$	$\min(\hat{\lambda}_n, \rho_{n-1} \times \delta_{\text{dec}})$	$\min(\hat{\lambda}_n, \rho_{n-1} \times \Delta_{\text{dec}})$

short-term congestion. The buffer queue length aims at a target minimal size at all times in accordance with the predicted network state, i.e. *target queue length*. Here, when we refer to the *n*th flow control point, we mean the time instant at which the *n*th period starts. Let

$\hat{\lambda}_n$: the desired sending rate;
ρ_n: the requested sending rate;
q_n^*: the target queue length at the *n*th flow control point;
q_n: the number of packets seen at the *n*th flow control point;
q_n^{v}: the *virtual* queue length at the *n*th flow control point;
$\hat{q}_n^{\text{v}}$: the estimate of the virtual queue length at the *n*th flow control point;
μ: the average packet playback rate;
$\tilde{\zeta}_n$: the amount of skipped packets in the *n*th flow control period;
$\tilde{q}_n^{\text{retx}}$: the amount of packet retransmission in the *n*th flow control period.

Two parameters are defined here regarding the control of the receiving buffer queue length: target queue length and *virtual queue length*. The target queue length represents the receiver's wish of how fast or slow the sender should send the packets in the next flow control period. This rate may be different from the requested sending rate—the rate requested by the receiver in the flow control message. In the case that the requested sending rate is less than the desired sending rate, selective transmission is performed at the sender. The sender gives higher transmission priority to more important packets, e.g. packets of I frames and retransmitted packets. As a result, less important frames (packets) are purposely skipped. The virtual queue length is the queue length if both selected and skipped packets were sent and received by the receiver.

3.1.1. Target queue length

R^3CP^+ supports packet retransmission. The key issue for real-time packet retransmission is that one must detect the loss of packets early enough—at least one RTT before its deadline to make retransmission effective. In our previous work [11], a minimal target queue length of the receiving buffer is derived which is one RTT-equivalent amount of data ($P_{\hat{d}_n}$). The simulation results showed that packet retransmission for real-time sessions is feasible and effective. By maintaining only this minimal amount of packets at the receiving buffer, one can achieve fairly good playback performance. The target queue length is given as follows:

$$q_n^* = P_{\hat{d}_n} + P_{\text{I\_distance}}, \tag{9}$$

where $P_{\text{I\_distance}}$ is the amount of packets in the retransmission window, i.e. the amount of packets checked in each retransmission control.

3.1.2. Virtual queue length

When the requested sending rate is less than the desired sending rate, selective transmission is performed. The amount of packets to skip $\tilde{\zeta}_n$ is given as follows:

$$\tilde{\zeta}_n = (\hat{\lambda}_n - \rho_n)\hat{d}_n + \hat{q}_n^{\text{retx}}. \tag{10}$$

Otherwise, $\tilde{\zeta}_n$ is 0. The virtual queue length is defined as:

$$q_n^{\text{v}} = q_n + \tilde{\zeta}_{n-1}. \tag{11}$$

To obtain the desired sending rate at the *n*th flow control point, the receiver makes a prediction of the number of packets that will be present in the buffer at the $(n+1)$th and $(n+2)$th flow control points. First, we have

$$\hat{q}_{n+1}^{\text{v}} = q_n^{\text{v}} - \mu\hat{d}_n + \rho_{n-1}(1 - \hat{p}_n)\hat{d}_n + \tilde{\zeta}_n. \tag{12}$$

The virtual queue length at the beginning of the $(n+1)$th flow control period is equal to the virtual queue length at the *n*th period less the number of packets removed (at the rate of μ) plus new arrivals during the period, and the packets skipped, if any. In the equation, the sender's sending rate is assumed to be the previous requested sending rate. Similarly, we have

$$\hat{q}_{n+2}^{\text{v}} = q_{n+1}^{\text{v}} - \mu\hat{d}_n + \hat{\lambda}_n(1 - \hat{p}_n)\hat{d}_n. \tag{13}$$

Here the sending rate is the desired sending rate to be computed.

We wish that at the beginning of the $(n+2)$th flow control period, the virtual queue length can at least meet the target queue length, i.e. $\hat{q}_{n+2}^{\text{v}} = q_{n+2}^*$. Rewriting Eqs. (12) and (13), the desired sending rate is obtained as follows:

$$\hat{\lambda}_n = \frac{P_{\text{I\_distance}} + 3\mu\hat{d}_n - q_n^{\text{v}} - \tilde{\zeta}_n}{(1 - \hat{p}_n)\hat{d}_n} - \rho_{n-1}. \tag{14}$$

3.2. Requested sending rate—storing extra data during unloaded state and avoiding congestion otherwise

The desired sending rate only represents the receiver's wish to the sender to ensure continuous playback of frames.

However, over the Internet, packet delivery performance is indeed dependent on the network load. The requested sending rate is determined according to the following algorithm where both the history and the forecast of the network condition are taken into account. Strategically, receiver uses different levels of rate increase and decrease in the determination of the requested sending rate depending the state of the network. The network is assumed to be in one of the following three states:

- 'UNLOADED' if $p_n < p_{\text{low}}$;
- 'LOADED' if $p_{\text{low}} < p_n < p_{\text{high}}$;
- 'CONGESTED' if $p_{\text{high}} < p_n$.

The scheme is summarized in Table 1. The algorithm for determining the requested sending rate is explained as follows:

Case 1 (The current transmission path is in the state of UNLOADED). If the current transmission path is in the state of UNLOADED, the flow control strategy is to store as many data as possible in the receiving buffer when the delivery condition is good. In other words, receiver will ask sender to increase its sending rate but with different amount of increase depending on the forecast of the future network load.

- If the forecast state is also UNLOADED, receiver makes a more aggressive attempt to increase the rate with a larger amount of increase Δ_{inc}. The final requested sending rate is bounded by the peak rate of the session $\hat{\lambda}_{\max}$.
- If the forecast state is LOADED and the current observation is UNLOADED, it is considered that the current measurement shows a sign of improvement of the network condition. However, the receiver will take a cautious step by increasing the sending rate with only a smaller amount denoted by δ_{inc}, $\delta_{\text{inc}} < \Delta_{\text{inc}}$.
- If the forecast is in the CONGESTED state, the current measurement is only considered as a signal of possible congestion relief; more observations are needed. Thus, the receiver takes the minimum of the current receiving throughput and the previous requested rate plus a smaller amount of increase.

Case 2 (The current transmission path is in the state of LOADED). If the current measurement indicates that the transmission path is in the LOADED state, the receiver will be conservative; the strategy is to retain the status in quo—no rate increase or decrease. The new requested sending rate takes the minimum of the new desired sending rate and the current sending rate (i.e. the previous requested sending rate). It means that if the new desired sending rate is less than the current sending rate, the receiver is not *greedy*; it only asks the sender to send the data at the rate necessary to maintain a reasonable smooth playback of the session even a large amount of bandwidth may be available. Adversely, if the current sending rate is smaller than the new desired sending rate, the receiver (or the session) uses the current sending rate as the requested rate. It is self-controlled rather than using the network bandwidth arbitrarily without considering others. The goal is to avoid congestion and be fair to all other types of traffic.

- If the forecast state is UNLOADED, the current measurement might show a transient behavior of the network or it could be a sign that the network load is increasing. The strategy is to retain the status in quo.
- If the forecast is also LOADED, the receiver infers that the network condition is stable and the strategy is to keep the status unchanged.
- If the forecast is CONGESTED, it is considered that with a great chance that the network load will get worse. As a result, receiver will just follow the previous requested rate.

Case 3 (The current transmission path is in the state of CONGESTED). If the current measurement of the transmission path is in the CONGESTED state, it is inferred as a sign of possible network congestion. The design rationale is to start congestion avoidance by reducing sending rate. Receiver will continuously ask the sender to decrease its rate until the situation is relieved. Depending on the forecast of the future state, different levels of multiplicative decrease of the currently measured throughput are taken.

- If the forecast is UNLOADED or LOADED, receiver will take a small reduction of the rate. Again the session is self-controlled—only the minimum of the desired sending rate and the reduced throughput is taken.
- If the forecast is CONGESTED, it is taken as an indication that the network situation will remain congested. Receiver takes a larger rate reduction action to relieve congestion.

In all cases, if the new desired sending rate is less than the reduced throughput, the receiver takes only what it needs to maintain the target queue length.

4. Performance evaluation

We have evaluated the proposed prediction-based flow/congestion control protocol both via simulations and actual Internet experiments. The protocol was implemented in a video player/browser system on top of the RTP/UDP protocols on Windows95. The simulations focused on the detailed analysis of the protocol behavior. Some of the Internet experimental results are presented here. There are several goals: (a) to show the improvement of the proposed scheme in maximizing the real-time playback performance

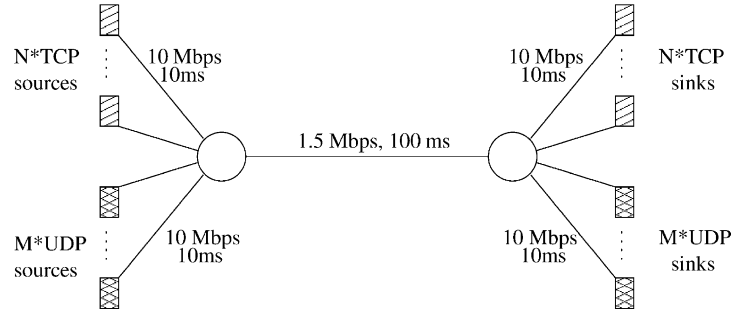
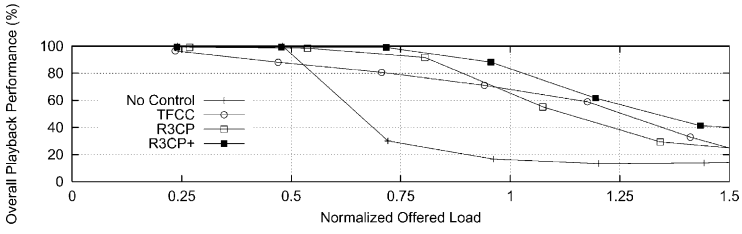
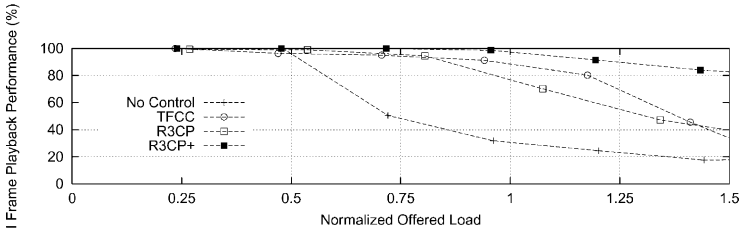


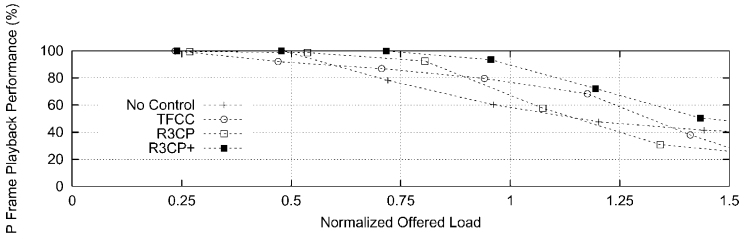
Fig. 2. Simulation configuration.



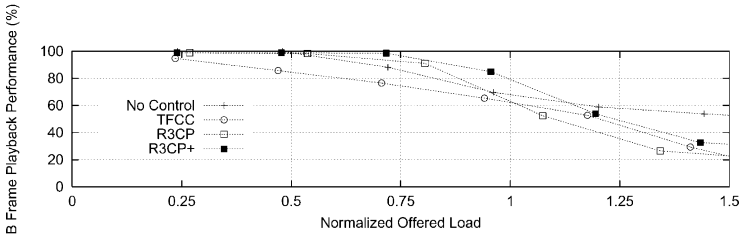
(a) overall playback performance



(b) I frames

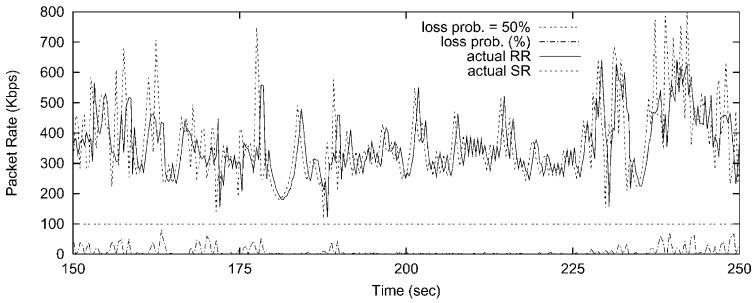


(c) P frames

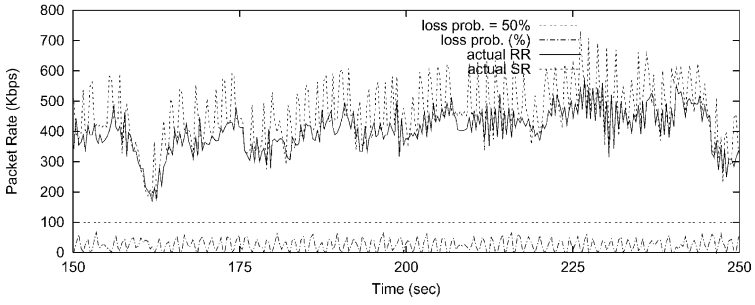


(d) B frames

Fig. 3. Comparison of the playback percentage of video frames. (a) Overall playback performance; (b) I frames; (c) P frames; (d) B frames.



(a) under R^3CP^+



(b) under $TFCC$

Fig. 4. Comparison of the packet sending rate, receiving rate and packet loss probability when the normalized offered load is 0.96. (a) Under R^3CP^+; (b) under TFCC.

of stored video over the best-effort Internet; (b) to understand how well the scheme can react to transient network load fluctuation and loss; (c) to demonstrate the effectiveness of the scheme in achieving fairness among real-time traffic while avoiding network congestion; and (d) to demonstrate the effectiveness of the scheme in achieving reasonably fair bandwidth sharing with TCP connections.

To provide some context, we compare the performance of R^3CP^+ with that of three other schemes: transmission without rate/congestion control, R^3CP [11] and the triple-feedback based congestion control (TFCC) scheme. In R^3CP, the sender transmits data at the desired sending rate. It exhibits the same behavior in bandwidth sharing as of typical UDP-based real-time flows—they grab as much bandwidth as they can. It is used as a baseline solely for comparison. TFCC employs loss-based congestion control and the popular additive increase/multiplicative decrease algorithm to control of rate adjustment, i.e.:

if $p_n < p_{low}$, *requested_rate* = *max*(*current_throughtput* + *INC*, *max_rate*);
else if $p_{low} < p_n < p_{high}$, *requested_rate* = *current_throughtput*;
else if $p_{high} < p_n$, *requested_rate* = *max*(*current_throughtput/DEC*, *min_rate*).

All simulations are performed by using ns [21]. The network configuration is shown in Fig. 2. We consider a single 1.5 Mbps congested link shared by a number of TCP connections and UDP/RTP-based video flows. We use an actual video trace (the 'Star War' movie [22]) as the video traffic source in the simulation. The first ten-minute section (14,400 frames and 28,120 packets) is used. Some important meta data of the simulated video stream are as follows: the nominal frame rate is 24 frames/s; the frame pattern is *IBBPBBPBBPBB*; hence, I_distance is 12. For transmissions which do not use any control mechanisms, in order to start the playback smoothly, the first I frame is transmitted with error recovery; once the first I frame has been successfully received, the playback begins immediately. The target queue length in all the experiments using R^3CP or R^3CP^+ is set to its *minimum* value as required by the look-ahead retransmission scheme.

4.1. Playback performance

In Fig. 3, we show the overall playback performance and the performances of individual frame types. The *playback performance* is defined as the percentage of frames that are successfully played back. In these experiments, an in-time completely received frame is not played if its referenced frame(s) is not present due to the inter-frame dependence. Each video connection has an average rate of 378 Kbps. From the figure, we can see that R^3CP^+ outperforms the other schemes in all ranges. This is mainly due to the benefit of employing congestion control with a prediction

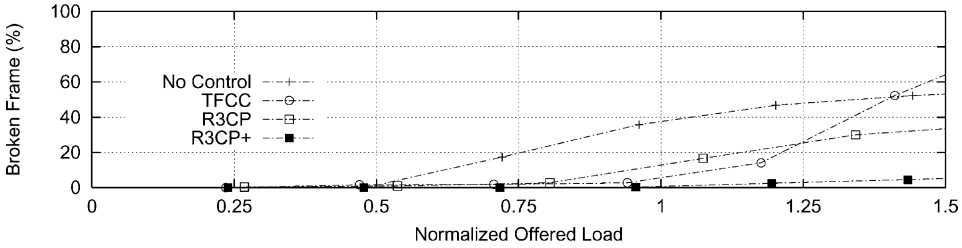


Fig. 5. Comparison of the percentage of broken frames.

mechanism and the in-time recovery of lost packets. R^3CP^+ uses more information to assess network states and, thus, better adapts the sender's sending behavior to the actual network conditions. It successfully avoids the possibility of congestion by not increasing the rate too quickly. Notably, the percentage of type I frames that are successfully played back remains higher than 80% in the range where the normalized offered load is over 0.75.

In Fig. 4, we show the histograms of the requested sending rate, receiving throughput and packet loss probability under R^3CP^+ and TFCC. The normalized offered load is 0.96. We can see that under R^3CP^+, the packet loss probabilities are much lower than those in TFCC. This is again because in R^3CP^+, the rate adjustment is based on not only newly sampled state information, but also on the history. This helps avoid unnecessary response to transient load changes and enhances the system stability. Moreover, one can see in Fig. 4(a) that under the fine-grained rate adjustment algorithm in R^3CP^+, the receiving throughput is close to the sending rate. On the other hand, in TFCC, the receiving throughput is much lower than the sending rate, which fluctuates wildly throughout the course. This results in a large number of packet loss (i.e. up to more than 20% loss ratio).

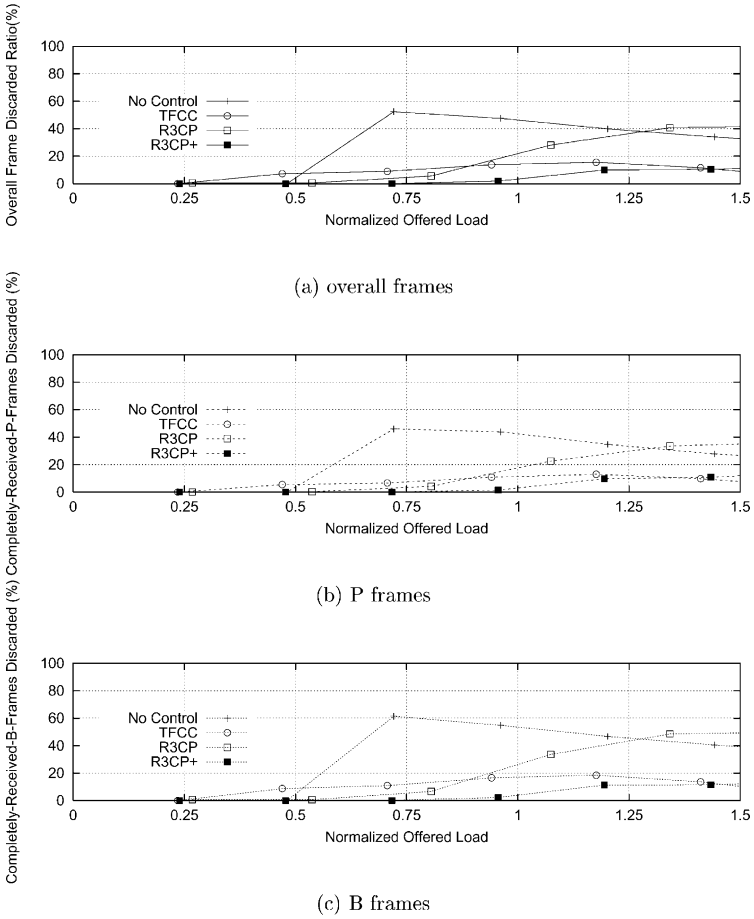
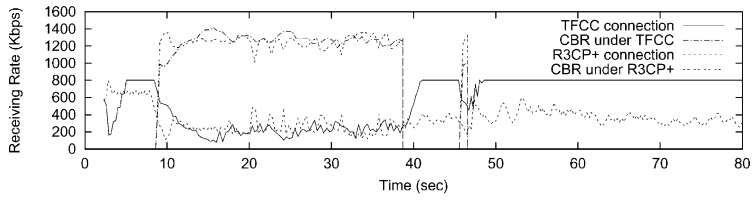
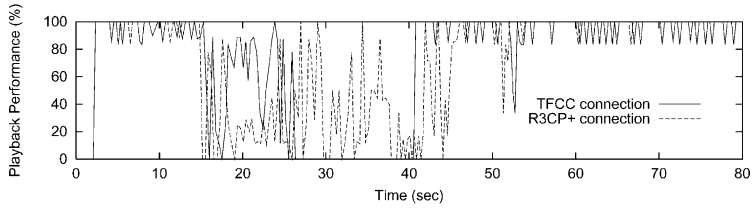


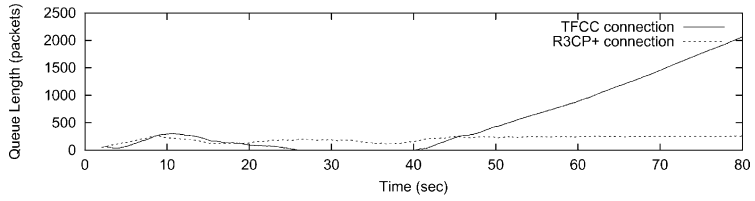
Fig. 6. Comparison of the total percentage of frames that are completely received but discarded due to the absence of the referenced frames. (a) Overall frames; (b) P frames; (c) B frames.



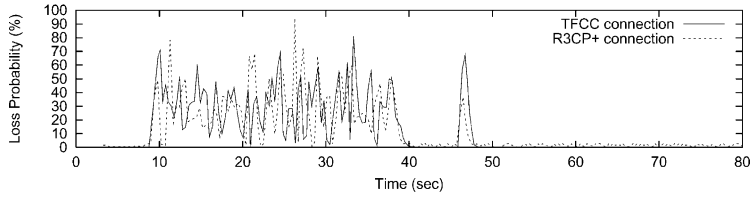
(a) packet receiving rate



(b) frame playback performance



(c) receiving buffer queue length



(d) packet loss probability

Fig. 7. Responsiveness to transient and long-lasting congestion events. (a) Packet receiving rate; (b) frame playback performance; (c) receiving buffer queue length; (d) packet loss probability.

Next, we analyze the performance between the frames received and the frames that can be successfully played out. For frames received, they can be distinguished in four categories: *broken* frames, *late* frames, *orphan* frames and *playable* frames. Because most of the frames are segmented into a number of UDP packets, a frame is said to be broken if any of its constituent packets are lost. Broken frames are not playable. In Fig. 5, we compare the percentage of broken frames when different control mechanisms are used. One can see that R^3CP^+ has the best performance. For those runs without any additional transmission control, they severely suffered from random packet loss. For R^3CP^+, the congestion control and the execution of selective transmission, the real-time session focuses its packet delivery on transmitting those most important frames given the limited bandwidth available to it, thus achieving the best performance.

Fig. 6 shows the comparison of the total percentage of frames that are completely received on the schedule but could not be played due to the loss of the referenced frames. By employing selective retransmission of lost packets, especially the important I frames, both R^3CP^+ and TFCC with R^3CP achieve better overall playback performance and bandwidth utilization.

4.2. Responsiveness and stability

In this section, we want to show the effectiveness of the proposed rate/congestion control scheme in handling long-lasting as well as transient congestion events. In the

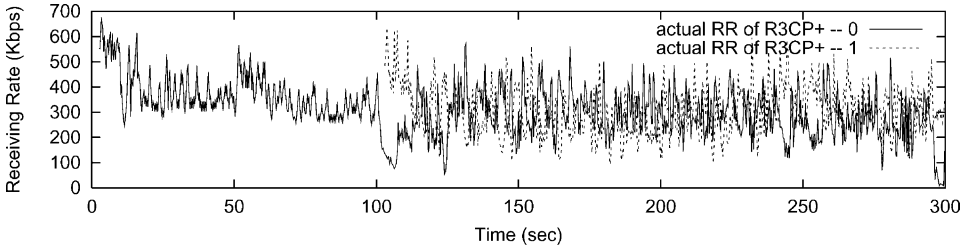
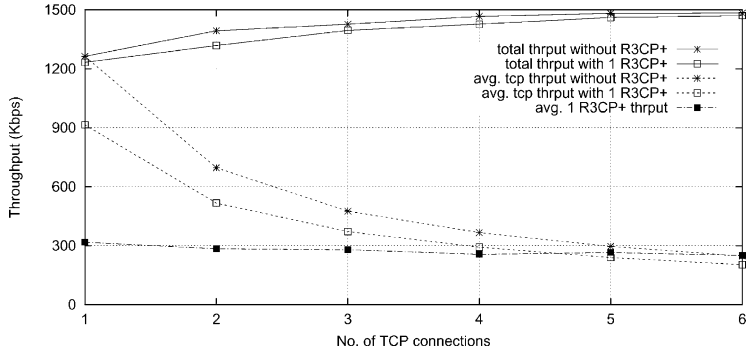
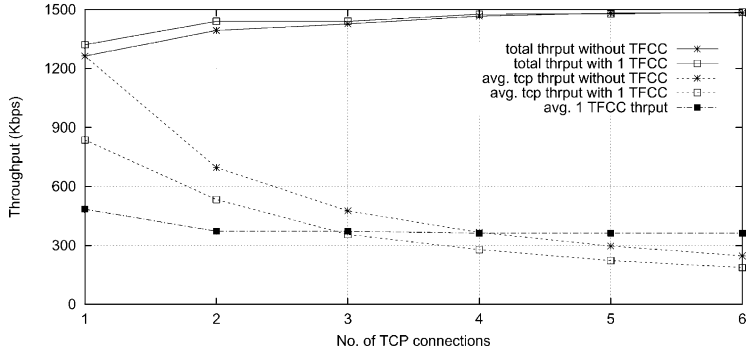


Fig. 8. Bandwidth sharing between R^3CP^+.

experiments, two background traffic are generated: a constant-bit-rate UDP flow which start to send fixed-size packets (1000 bytes) at a rate of 1.454 Mbps from time 8.5 to time 38.5; and a burst of fixed-size UDP packets (1000 bytes) which is injected into the congested link at a rate of 1.454 Mbps from time 45.4 to time 46.4. We compare the usage of various resources of a real-time flow under R^3CP^+ or TFCC. Playback of the video stream starts at time 2. Fig. 7 shows the histograms of the packet receiving rate and the corresponding frame playback performance, queue length and measured packet loss rate probability. First, we can see that the flow using R^3CP^+ scheme converges to a steady state faster than does that under the TFCC scheme. Notice that under TFCC, the flow continues to grab available bandwidth until it reaches its upper bound, which is set to 800 Kbps. At time 8, congestion occurs. The adaptive congestion control mechanism in R^3CP^+ is able to compete with the background UDP traffic during the period of congestion and maintains a number of packets in the receiving buffer so to ensure continuous playback of video frames. On the other side, TFCC fails to contend for bandwidth with aggressive UDP traffic. From time 20 to time 40, no frames are played under TFCC. In terms of the playback performance (in Fig. 7(b)), R^3CP^+ performs much better than TFCC. When congestion ended at time 38, both R^3CP^+ and TFCC pick up the bandwidth rapidly. Note that R^3CP^+ converges to the rate it needs (around 380 Kbps) while TFCC hoards the



(a) R^3CP^+ flow



(b) $TFCC$ flow

Fig. 9. Bandwidth sharing between congestion-controlled real-time flows and TCP connections. (a) R^3CP^+ flow; (b) TFCC flow.

Table 2
Statistics of the round-trip delay and loss probability

Source	Avg. RTT (ms)	Loss Prob.
NTU	10.205	0.1267
NCTU	32.378	0.0522
Stanford Univ.	258.891	0.00933

bandwidth, resulting in continuous growth of the buffer queue length.

4.3. Fairness with real-time sessions

In Fig. 8, we study how real-time flows employing R^3CP^+ control mechanisms share link bandwidth. First, R^3CP^+ flow starts at time 0.33, represented by the dark line. Another R^3CP^+ flow joins at time 100.19, represented by the dash line. A background UDP flow is always up which sends packets at a rate of 909.09 Kbps. The link capacity is 1.5 Mbps. One can see that after the second flow enters the congested link, the first flow detects this situation and reduces its rate; in a very short period of time, two flows reach almost the same throughput and share the link bandwidth equally.

4.4. Fairness with TCP traffic

In this section, we will study the bandwidth shared between R^3CP^+ flows and TCP connections. definition of TCP 'friendliness' has been widely adopted in the congestion control of real-time applications [12–15]. In essence, if a UDP connection shares a bottleneck link with TCP connections of the same link, then the UDP connection should receive the same share of bandwidth (i.e. achieve the same throughput) as does a TCP connection. A concern is raised with this approach. That is, real-time applications have more stringent performance requirements than do non-real time applications (those mainly using TCP). While equally sharing the bandwidth over a bottleneck link might sound fair at the first glance, we believe that relative fairness between UDP and TCP traffic is only achievable if a congestion control algorithm for real-time applications can be shown to be 'reasonably fair' to TCP. This philosophy is more practical to meet the performance objectives of both types of applications as well as to maximize link utilization. Nevertheless, one still needs to be careful about how much more bandwidth should be given to a real-time connection, which is still an open research issue [23]. In the following, we will show that the proposed real-time flow/congestion control scheme only allows a real-time connection to obtain a slightly larger amount of resources than can TCP during the congestion.

Table 3
Comparison of frame playback performance

Source	R^3CP^+ (%)	R^3CP (%)	No control (%)
NTU	91.25	90.63	9.55
NCTU	90.24	93.36	8.27
Stanford Univ.	96.23	92.38	59.16

Let us look at the average throughput a TCP connection can obtain when there are multiple TCP connections transmitting over a shared link. In Fig. 9(a), one can see that the average throughput continues to decline when more TCP connections join in. Now, consider the addition of a R^3CP^+ connection. When there is no congestion, the R^3CP^+ connection is very self-controlled because it adjusts its rate without aggressively increasing it and causing congestion. It takes only what it needs and shares bandwidth with TCP connections in a consistent and friendly manner. When more TCP connections are added to the link, the network becomes congested. The R^3CP^+ connection executes its congestion control and reduces the rate. There is no 'TCP bandwidth starvation' situation. For TCP connections, the average throughput is reduced a bit less than is the case without the R^3CP^+ connection. The effect on the throughput reduction for TCP connections is mainly due to the competition among them with or without the R^3CP^+ connection. In Fig. 9(b), we again show results of the same experiments using TFCC. One can see that the connection using TFCC obtains more bandwidth than do the connection using R^3CP^+. Hence, the average throughput of TCP connections with the R^3CP^+ connection is greater than that with the TFCC connection.

4.5. Internet experiment results

Here, we present some of the performance results of the experiments that were conducted over the Internet. Three different Internet path environments are tested, each representing a different transmission characteristic. A video server is located at each of the three sites (bigzoo.g1.ntu.edu.tw, fanga.iim.nctu.edu.tw and deepblue2.standord.edu). The client station is located at the computer center of the National Taiwan University (140.112.3.120). Table 2 gives a snapshot of the RTT and loss probability statistics of the three sets of the experiments. In Table 3, we compare the frame playback performance of the runs under R^3CP^+, R^3CP and transmission without any control. Detailed results are reported in Ref. [24].

5. Conclusions

In this paper, we have presented a prediction-based flow/congestion control scheme called R^3CP^+ for real-time stored packet video transfer over the best-effort based Internet. In this protocol, several control mechanisms are used. First, an adaptive filter is used to predict the network state based on the current measurement as well as the history. The objective is to capture both the 'trend' and the 'transiency' of the network load behavior. With the forecast, the receiver calculates its desired sending rate by taking into account the occupancy of the receiving buffer. In

this way, we ensure that the receiver is self-controlled, and that it does not ask the sender to send more data than it needs to assure its continuous playback of frames. Finally, depending upon the different assessments of the network condition, the receiver determines the requested sending rate differently.

We have studied and compared the performance of the proposed R^3CP^+ protocol with other schemes via simulation and through actual implementation. The simulation results show that R^3CP^+ outperforms schemes like the popular linear-increase/multiplicative-decrease algorithm in all ranges. It is due to the use of adaptive filter in network state forecast and assessment as well as finer control of rate adjustment. Thus, it cannot only better adapt the packet sending rate to the actual condition, but also avoid unnecessary response to transient load changes. This indeed helps to maintain system stability. The combined use of a buffer-occupancy based desired sending rate with in-time packet retransmission and selective transmission also significantly improves the playback performance.

We have also studied the issue of resource sharing between R^3CP^+ flows as well as with TCP connections. The results show that because each R^3CP^+ flow is congestion-conscious when it performs rate adaptation, it can achieve fair bandwidth sharing at a bottleneck link. Moreover, a R^3CP^+ flow can also share bandwidth with TCP connections in a reasonably friendly fashion. By reasonably friendly, we mean that the real-time connection only takes a slightly larger amount of bandwidth than does TCP throughput during congestion. The reason for the throughput reduction for TCP connections is mainly the competition among them with or without connection. Therefore, there is no TCP bandwidth starvation situation when real-time connections using the flow/congestion control mechanisms of R^3CP^+.

References

[1] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, Resource ReSerVation Protocol (RSVP)—Version 1 functional specification, RFC 2205, September 1997.

[2] J. Wroclawski, The use of RSVP with IETF integrated services, RFC 2210, September 1997.

[3] J. Wroclawski, Specification of the controlled-load network element service, RFC 2211, September 1997.

[4] S. Shenker, C. Partridge, R. Guerin, Specification of guaranteed quality of service, RFC 2212, September 1997.

[5] S. Chong, S-Q. Li, J. Ghosh, Predictive dynamic bandwidth allocation for efficient transport of real-time VBR video over ATM, IEEE JSAC January (1995).

[6] J.M. McManus, K.W. Ross, Video-on-demand over ATM: constant-rate transmission and transport, IEEE JSAC August (1996).

[7] M. Grossglauser, S. Keshav, D.N.C. Tse, RCBR: a simple and efficient service for multiple time-scale traffic, IEEE/ACM Transactions on Networking December (1997).

[8] A.M. Adas, Using adaptive linear prediction to support real-time VBR video under RCBR network service model, IEEE/ACM Transactions Networking October (1998).

[9] Z. Chen, S. Tan, R. Campbell, Y. Li, Real time video and audio in the World Wide Web, Fourth International World Wide Web Conference, 1995.

[10] C. Papadopoulos, G.M. Parulkar, Retransmission-based error control for continuous media applications, Proceedings of NOSSDAV'96, 1996.

[11] Y. Sun, F.M. Tsou, M.C. Chen, A buffer-occupancy based adaptive flow control scheme with packet retransmission for stored video transport over Internet, IEICE Transactions on Communications November (1998).

[12] D. Sisalem, H. Schulzrinne, The loss-delay based adjustment algorithm: a TCP-friendly adaptation scheme, Proceedings of NOSSDAV'98, July 1998.

[13] R. Rejaie, M. Handley, D. Estrin, RAP: an end-to-end rate-based congestion control mechanism for realtime streams in the Internet, INFOCOM'99, 1999.

[14] J. Padhye, J. Kurose, D. Towsley, R. Koodli, A TCP-friendly rate adjustment protocol for continuous media flows over best effort networks, ACM Sigmetrics'99, May 1999.

[15] S. Floyd, K. Fall, Router mechanisms to support end-to-end congestion control, Technical Report, February 1997.

[16] D. Sisalem, Fairness of adaptive multimedia applications, ICC'98, 1998.

[17] S. Haykin, Adaptive Filter Theory, Prentice-Hall, Englewood Cliffs, 1986.

[18] D.D. Clark, D. Tennenhouse, Architectural considerations for a new generation of protocols, ACM SIGCOMM'90, September 1990.

[19] ISO/IEC International Standard 11172, Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbits/s, November 1993.

[20] ISO/IEC International Standard 13818, Generic Coding of Moving Pictures and Associated Audio Information, November 1994.

[21] UCB/LBNL/VINT, Network Simulator-ns (version 2), http://www-mash.cs.berkeley.edu/ns/.

[22] M.W. Garrett, A. Fernandez, MPEG-1 Video Trace, Bellcore, ftp://thumper.bellcore.com/pub/vbr.video.trace/MPEG.data, 1992.

[23] F. Kelly, Charging and rate control for elastic traffic, European Transactions on Telecommunications 8 (1997).

[24] I-K. Tsay, An implementation and performance evaluation of end-to-end joint flow/congestion control scheme for real-time stored packet video over Internet, Master Thesis, June 1999.