ESTIMATING PACKET ARRIVAL TIMES IN BURSTY VIDEO APPLICATIONS

Ali C. Begen and Yucel Altunbasak

School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA USA

{acbegen, yucel}@ece.gatech.edu

ABSTRACT

In retransmission-based error-control methods, the most fundamental yet the paramount problem is to determine how long the sender (or the receiver) should wait before deciding that an unacknowledged (or a missing) packet is lost. This waiting time is generally referred to as retransmission timeout (RTO). An accurate RTO estimation has two main advantages: First, the lost packets can be identified earlier, and hence, can be recovered faster. Second, redundant retransmissions can be avoided, which subsequently not only saves the network resources, but also helps existing network congestion alleviate sooner. Although it is statistically possible to prevent any unnecessary retransmission at the expense of long error-recovery times, such an approach can only be justified for data applications; it is not well-suited for delay-sensitive applications, for which the agility in recovering the lost packets is as important. With this motivation, we recently introduced an RTO estimation algorithm for delay-sensitive applications [1]. Provided that the packets are transmitted at equal intervals, this technique successfully estimates the arrival times based on the interarrival-time observations. In this study, we relax the requirement of equal transmission intervals and generalize our technique to handle bursty video applications.

1. INTRODUCTION

Because of its best-effort nature, any packet injected to the Internet is subject to loss or random delay. Naturally, it is safe to assume that a missing (or an unacknowledged) packet is lost, if it has been a long time since its transmission. In TCP jargon, this waiting time is referred to as retransmission timeout (RTO). However, because of the potential delay jitter, it is important to employ a sufficiently large RTO in order to allow late packets to eventually arrive. While employing a large RTO value usually has a minimal effect on data applications, unfortunately, delay-sensitive applications do not have the luxury of overwaiting. Decisions on missing packets should be made quickly so that well-timed actions can be taken against the lost ones. In our recent work [1], we proposed an efficient technique for timely inference of the late/lost packets in streaming applications. Assuming that the server transmitted the packets at equal intervals, this technique utilized interarrival times to estimate the arrival times of incoming packets. Internet experiments showed that our approach provided significant improvements in the streaming quality while keeping the redundant retransmission rate at a negligible level.

In low-delay video streaming applications, it is a common practice to transmit the video frames as soon as they are packetized in order to avoid unnecessary delays. However, because of the efficient predictive-coding techniques that are an integral part of the popular video coding standards, *e.g.*, MPEGx and H.26x, each encoded video frame differs in size and potentially produces a different number of packets. Furthermore, when streaming at high bitrates, even the smallest video frame may not fit into one packet. If all packets belonging to a particular frame are transmitted back-to-back, the video traffic inevitably becomes bursty. Combined with the delay jitter experienced along the path, the varying nature of the transmission times causes the video packets arrive at the client at less-predictable times, which consequently renders the RTO estimation more difficult. In this study, we address this problem and propose a generalized RTO estimation algorithm that can handle bursty video applications. With both simulations and Internet experiments, we evaluate the performance of this *burst-aware* RTO estimator.

RTO estimation has long been studied for TCP. To date, several enhancements (*e.g.*, [2, 3, 4]) have been proposed to improve the initial implementation [5]. However, these proposals are naturally not suitable for delay-sensitive applications in the sense that their estimators are excessively conservative and adapt to the network conditions slowly. On the other hand, [6, 7, 8] consider different retransmission schemes for real-time streaming. Among these studies, the most comprehensive work is [8], where Loguinov and Radha evaluate several round-trip time based RTO estimators for very low-bitrate video streaming. In contrast to [8], our estimator [1] uses interarrival times for RTO estimation, which makes it more responsive and precise in high-bitrate streaming applications.

In the rest of the paper, we first introduce the terminology and the basics of the RTO estimation in Section 2. In Section 3, we continue with the details of the proposed approach and its comparison with other RTO estimators. We present the results produced from Internet video streaming experiments in Section 4. Finally, we conclude the paper in Section 5.

2. PRELIMINARIES

2.1. Terminology

One of the major differences between the conventional RTO estimators and our approach is that the latter is entirely implemented at the client side. In this scheme, the server does not carry out any computation; it merely transmits the video packets and responds to the requests received from the client. In our analysis, we will refer to each packet with a unique number, l, which can be associated with the *Sequence Number* field in the RTP header [9]. We denote the transmission times (at the server), arrival times (at the client)

0-7803-9332-5/05/$20.00 ©2005 IEEE

and decoding deadlines by $t_T(l)$, $t_A(l)$ and $t_D(l)$, respectively. Note that $t_D(.)$ is common for all packets belonging to a particular video frame. In addition, different frames may share the same decoding deadline if the decoding order of the frames is different than their display order.

The transmission times of the video packets depend on several factors. Namely, the number of frames in a group of pictures (GOP), the GOP structure, encoding bitrate, video frame rate and IP packet size are the main parameters that vary the transmission times of the packets in a GOP. As an illustrative example, consider Fig. 1, where the GOP structure consists of one I-frame and nine P-frames. Assume that the video is streamed at 20 f/s, and the I-frame and each of the P-frames produce seven and two IP packets, respectively. Let $\Delta t_T(l)$ represent the intertransmission time of packet l, which is defined as

$$\Delta t_T(l) = t_T(l) - t_T(l-1). \tag{1}$$

The mean intertransmission time can be computed by using

$$\overline{\Delta t_T} = \frac{\text{GOP Duration}}{\text{\# of Packets in a GOP}}, \tag{2}$$

which is 20 ms in our example. However, as shown in Fig. 1, Δt_T can be as low as a few milliseconds, and as high as 50 ms. It is clear that $\overline{\Delta t_T}$ is largely inadequate to define the transmission regime of the server.

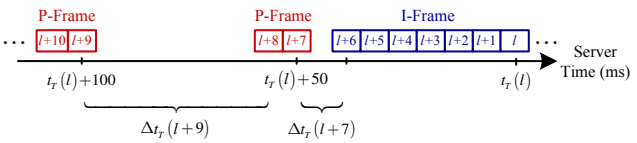

Fig. 1. The variation of intertransmission times at the server side.

As mentioned above, the variation in the intertransmission times depends on many factors. Fortunately, our tests with a standard H.264 video codec show that this variation follows a similar pattern for consecutive GOPs, provided that encoder and video-specific parameters are kept the same. However, in some configurations, the pattern breaks because of a sudden scene change. Nevertheless, without loss of generality we can safely assume that the pattern for the first GOP is also valid for the subsequent GOPs, and the server conveys any new pattern information to the client when there is a change in the encoding/packetization process.

2.2. Timeout Mechanism

Similar to intertransmission times, we define interarrival times at the client. The interarrival time of packet l is equal to the amount of the time passed since the arrival of the previous packet. If we denote the index of the last received packet before packet l by l^*, then the interarrival time for packet l is given by

$$\Delta t(l) = t_A(l) - t_A(l^*). \tag{3}$$

For non-received packets, we clearly have $t_A(l) = \infty$ and $\Delta t(l) = \infty$. The main idea behind our RTO estimation algorithm is to estimate the subsequent interarrival time and project the corresponding arrival time after each packet arrival. We use the notation of $\widetilde{\Delta t}(.)$ and $\widetilde{t_A}(.)$ to denote the estimated interarrival and projected arrival times, respectively. The estimation is based on the latest interarrival time and Δt_T. That is,

$$\widetilde{\Delta t}(l+1) = f(\Delta t(l), \Delta t_T(l+1)), \tag{4}$$

for some function f. For example, in Fig. 2 packet $l+1$ is expected to arrive by $\widetilde{t_A}(l+1)$, which is computed by

$$\widetilde{t_A}(l+1) = t_A(l) + \widetilde{\Delta t}(l+1). \tag{5}$$

In case of packet $l+1$ does not arrive within $\widetilde{\Delta t}(l+1)$ time units, then the client presumes that this packet is lost and times out.

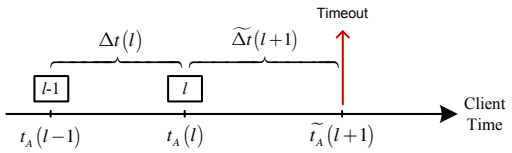

Fig. 2. Timeout mechanism.

2.3. Performance Metrics

We evaluate the performance of our RTO estimator with two metrics. First, we define p_f as the estimation failure probability. This indicates the rate of misidentifying non-lost packets as they are lost. It can be computed by using

$$p_f = P\{\widetilde{t_A} < t_A\}, \text{ for } t_A < \infty. \tag{6}$$

The second metric is the average overwaiting time. The excessive waiting time for a non-lost packet l is defined as

$$w(l) = \begin{cases} \widetilde{t_A}(l) - t_A(l), & \text{if } t_A(l) < \widetilde{t_A}(l) < \infty; \\ 0, & \text{if } \widetilde{t_A}(l) \le t_A(l) < \infty. \end{cases} \tag{7}$$

On the other hand, the waiting time spent for a lost packet can be computed by using a virtual arrival time extrapolated from the latest packet arrival time. If l^* denotes the index of the last received packet, the excessive waiting time for a lost packet is given by

$$w(l) = \widetilde{t_A}(l) - \left(t_A(l^*) + \sum_{l^* < l' \le l} \Delta t_T(l') \right), \text{ if } t_A(l) = \infty. \tag{8}$$

Naturally, there is a trade-off between p_f and average excessive waiting time (denoted by $\overline{w}$). Estimation failures can be largely avoided, if the client can tolerate a prolonged amount of time before timing out. As this ability diminishes, the client starts giving wrong decisions and may identify late packets as they are lost. Our goal is to reduce $\overline{w}$ while keeping p_f below a desired value.

3. RTO ESTIMATION

In this section, we study the details of the proposed RTO estimation algorithm. To this effect, we benefited from *ns*-2 network simulator [10] and Georgia Tech Internetwork Topology Models [11]. We developed an *ns*-2 server application that streams a pre-encoded video over RTP [9] and a client application that carries out RTO estimation. In particular, we used the test sequence FOREMAN (352×288), encoded with a H.264 encoder at a bitrate of 600 Kbps and frame rate of 20 f/s. The GOP structure consisted of one I-frame and nine P-frames, where each I-frame and P-frame was packed into seven and two IP packets, respectively. Our analysis of non-bursty video packet traces in [1] previously showed that forward-trip times had a larger variation compared to the interarrival times. Not surprisingly, Fig. 3 shows that this observation still holds, even if the video packets are injected into the network in a bursty manner. Fig. 3 also shows that the interarrival times

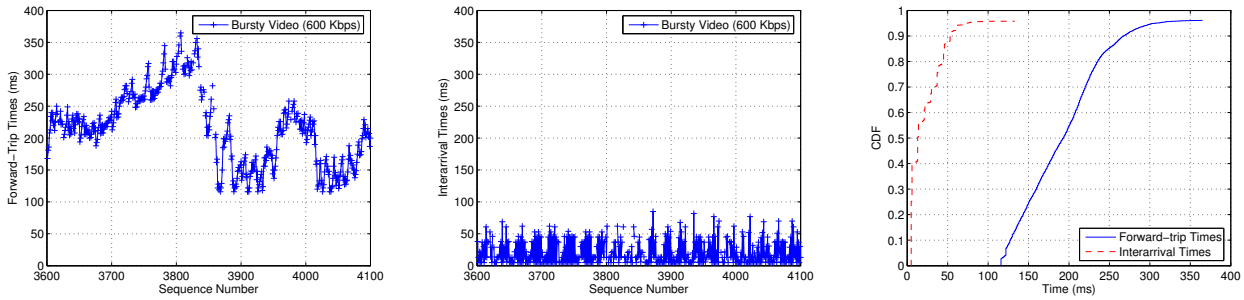

Fig. 3. Variation of forward-trip times (on the left), interarrival times (in the middle), and their distributions (on the right).

are mostly confined within a small region, although they present a noisy behavior due to the background TCP/UDP flows.

In continuous-media applications, a client successively receives packets and can easily extract the variation in the forward-trip times, provided that the client is aware of the intertransmission times. However, the exact forward-trip times cannot be computed without the knowledge of the transmission times at the server and a synchronization between the server and client clocks. Fortunately, to estimate the subsequent arrival time, we neither need to know the transmission times, nor require a clock synchronization. Based on (5), we merely need to estimate the interarrival time of the expected packet in order to compute its projected arrival time.

3.1. Burst-Aware RTO Estimator

In this section, we generalize the RTO estimation algorithm previously studied in [1]. The proposed burst-aware RTO estimator has the ability to run with any video traffic regime. Specifically, in this study we propose the following estimator:

$$\widetilde{\Delta t}(l+1) = \max\left(\Delta t_T(l+1), \alpha \times \Delta t(l) + \beta \times \Delta t_T(l+1)\right), \quad (9)$$

where α and β are some constants that determine the responsiveness of the estimator. For example, with $\alpha = 0$ and $\beta = 1$ we get an extremely aggressive estimator, which achieves a small average excessive waiting time ($\overline{w}$) but potentially a high failure probability (p_f). Using a larger β value may lower p_f, however, will inevitably increase $\overline{w}$. Recall that the variation between consecutive interarrival time samples is bounded except some impulsive points. If we ignore these impulsive points for the time being, a particularly successful estimator can be achieved with $\alpha = 7/8$ and $\beta = 7/8$ for bursty videos. We observe that (9) can closely track the actual arrival times and achieve a small $\overline{w}$ with these parameters. However, it is highly susceptible to sudden delay increases. In particular, the observed average excessive waiting time is 19 ms and the failure probability is 24.0%. We address this problem next.

3.2. Detecting Excessively-Delayed Packets

During the course of streaming, it is possible that two consecutive video packets are interleaved with several other packets belonging to different flows. In such cases, the latter packet can be excessively delayed and may be identified as it is lost (*e.g.*, see packets #3872 and #3935 in Fig. 3). One straightforward way to avoid such misidentifications is to increase the values of α and/or β. However, since such impulsive delay increases occur rarely, we rather keep α and β unchanged and use a supplementary timer only when it is needed. This secondary timer is called *latePacketTimer*. The client starts *latePacketTimer* when the expected packet does not arrive by the initial projected arrival time. If *latePacketTimer* also expires, the packet is registered as lost and the client times out.

A sketch of *latePacketTimer* is given in Fig. 4. Note that *latePacketTimer* defers the estimated arrival time for not only the expected packet but also all subsequent packets, since the proposed RTO estimator operates on the interarrival times rather than the arrival times. Also note that once *latePacketTimer* is started, the client does not start a second one until a new packet arrives. As opposed to exponentially backing off [12], this strategy allows to keep overwaiting time considerably shorter without sacrificing the accuracy.

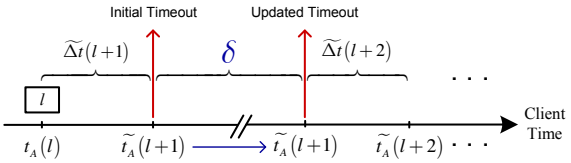

Fig. 4. Illustration of *latePacketTimer*.

In the implementation of *latePacketTimer*, the initial value of δ can be set to an arbitrary value since it is revised every time the timer is used. Its value is updated only if the received packet is not the expected packet or it is the expected packet but it arrived after its estimated arrival time. Let l_p and l_n be the indexes of the last packet received before *latePacketTimer* is started and the first packet received after *latePacketTimer* is started, respectively. We set the value of δ as follows:

$$\delta = \begin{cases} \min\left(\delta_{max}, t_A(l_n) - t_A(l_p)\right), & \text{if } l_n > l_p + 1 \text{ or} \\ & \quad t_A(l_n) > \widetilde{t_A}(l_n); \\ \delta, & \text{ow.} \end{cases} \quad (10)$$

Here δ_{max} represents the upper limit for δ. In this study, we observed that $3 \times \overline{\Delta t_T}$ is a good choice for δ_{max}.

With the introduction of *latePacketTimer*, the RTO estimation failure probability reduces from 24.0% to 0.3% at the expense of 7 ms increase (from 19 ms to 26 ms) in the average excessive waiting time. On the other hand, the burst-unaware RTO estimator [1] (obtained by replacing $\Delta t_T(.)$ with $\overline{\Delta t_T}$ in (9)) achieves a failure probability of 0.9% and an average excessive waiting time of 33 ms. This inferior performance stems from the fact that the burst-unaware RTO estimator has not been particularly tailored to handle bursty traffic. Finally, the enhanced TCP-like RTO estimator (see [1] for details) fails in 1.1% of the packets, although it

results in an average excessive waiting time of 118 ms. We compare all three RTO estimators in Fig. 5. The results clearly show that we can boost the performance of RTO estimation and subsequently the error-control/protection capability of the application by incorporating the packet transmission strategy of the server into (9).

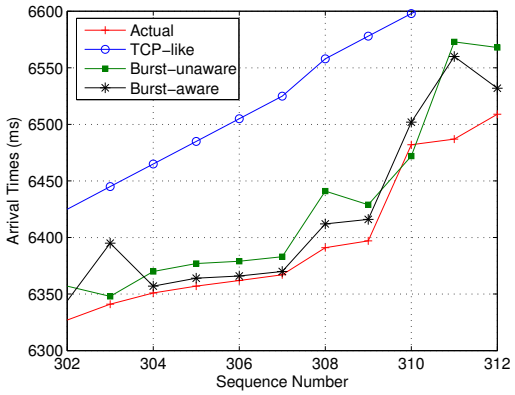

Fig. 5. Actual and estimated arrival times for three different RTO estimators.

4. EXPERIMENTAL RESULTS

We established an experimental platform in the Internet in order to assess the performance of different RTO estimators. On this platform, we ran a real-time video streaming application over RTP between a broadband client in Konya, Turkey and a server connected to Georgia Tech's campus network. The client simultaneously streamed real-time video and carried out the RTO computation. When a packet was identified to be lost, a retransmission request was sent to the server. Upon receiving the request, the server immediately retransmitted the requested packet.

We conducted our experiments in three sessions of 30 minutes, to evaluate all three RTO estimators. After each session, the mean delay and packet loss rate were measured to ensure that similar network characteristics were observed in all sessions. (The mean round-trip delay and mean one-way packet loss rate were approximately 250 ms and 6.0%, respectively.) We used a standard H.264 codec to encode the test sequence FOREMAN (352×288) at 600 Kbps with a frame rate of 20 f/s. In order to compensate for one-way delay jitter and produce some time for retransmissions, we employed a playout delay of 500 ms based on the initial observations. The packets that still could not be delivered by their decoding deadlines were not displayed, although each received packet was still used to decode subsequent predictively-coded frames. The results are summarized in Table 1.

Without any retransmission, the video quality severely suffers from missing packets. When an enhanced TCP-like RTO estimator is used, the client receives barely 1% more packets on time and improves the video quality by 1.3 dB. At the same time the average streaming rate increases by 7.5% due to the retransmissions, 19.1% of which are redundant. The relatively small improvement at the expense of a 7.5% rate increase indicates that majority of the retransmitted packets were late for decoding. In contrast, the burst-unaware RTO estimator delivers 3.8% more packets on time, while increasing the average streaming rate by 7.1%. On the other hand, the burst-aware RTO estimator delivers 0.1% more packets on time compared to the burst-unaware RTO estimator, which corresponds to 0.1 dB superior video quality. The burst-aware RTO estimator is also more network-friendly as it reduces the percentage of the redundant retransmission requests from 15.1% to 5.3%. The corresponding increase in the average streaming rate is 6.3%, which is smaller than those of both enhanced TCP-like and burst-unaware RTO estimators.

	% of Successful Packets	Average Quality	% of Redundant Ret. Requests
No ARQ	94.1%	34.3 dB	N/A
TCP-like	95.1%	35.6 dB	19.1%
Burst-unaware	97.9%	38.2 dB	15.1%
Burst-aware	98.0%	38.3 dB	5.3%

Table 1. Experimental results for the FOREMAN sequence.

5. CONCLUDING REMARKS

We studied a generalized RTO estimation algorithm that can be used within delay-sensitive applications producing any type of bursty packet traffic. An important application of our approach is that it can be integrated with the recently proposed rate-distortion optimized streaming techniques. This integration enables packet scheduling algorithms to give better decisions without requiring complex calculations that are peculiar to existing implementations. This way, we can achieve a more *reliable* and *realizable* real-time rate-distortion optimized streaming. In our future work, we will investigate into this integration further.

Acknowledgments

This work is supported by NSF under NSF award CCF-0430907.

6. REFERENCES

[1] A. C. Begen and Y. Altunbasak, "Timely inference of late/lost packets in real-time streaming applications," in *Picture Coding Symp. (PCS)*, 2004.

[2] M. Allman and V. Paxson, "On estimating end-to-end network parameters," in *ACM SIGCOMM*, 1999.

[3] L. Ma, G. R. Arce, and K. E. Barner, "TCP retransmission timeout algorithm using weighted medians," *IEEE Signal Processing Lett.*, vol. 11, no. 6, pp. 569–572, June 2004.

[4] R. Ludwig and K. Sklower, "The eifel retransmission timer," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 3, pp. 17–27, 2000.

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

[6] C. Papadopoulos and G. M. Parulkar, "Retransmission-based error control for continuous media applications," *ACM NOSSDAV*, 1996.

[7] I. Rhee, "Error control techniques for interactive low bitrate video transmission over the internet," in *ACM SIGCOMM*, 1998.

[8] D. Loguinov and H. Radha, "On retransmission schemes for real-time streaming in the internet," in *IEEE Int. Conf. Computer Communications (INFOCOM)*, 2001.

[9] RTP: A Transport Protocol for Real-Time Applications. [Online]. Available: http://www.ietf.org/rfc/rfc1889.txt

[10] S. McCanne and S. Floyd. Network simulator. [Online]. Available: http://www.isi.edu/nsnam/ns

[11] E. W. Zegura, K. L. Calvert, and M. J. Donahoo, "A quantitative comparison of graph-based models for Internet topology," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 770–783, 1997.

[12] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," in *ACM SIGCOMM*, 1987.