

TIMELY INFERENCE OF LATE/LOST PACKETS IN REAL-TIME STREAMING 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 delay-sensitive real-time streaming applications, when a packet is not delivered within an expected amount of time, an important issue is to determine whether the missing packet is merely delayed or is already lost. Historically, inherited from TCP jargon, the duration after which the missing packet is assumed to be lost is referred to as retransmission timeout (RTO). Regardless of doing a retransmission or not, timely inference of late/lost packets is crucial for real-time streaming applications, so that well-timed actions can be taken by the server/client. In this paper, we introduce a new client-driven RTO estimation algorithm that can be used in conjunction with any error-control method for streaming real-time audio/video. Our goal is to identify the lost packets as quickly as possible, while keeping the chance of any misidentification, *i.e.*, identifying a late packet as a lost packet, low. First, we study the dynamics of the RTO estimation process and demonstrate the steps involved in developing the proposed algorithm. Second, we combine our algorithm with a simple retransmission-based error-control method to analyze the impact of the RTO estimation in the context of a video streaming application. By Internet experiments, we show that our approach can substantially improve the video quality delivered to the clients without requiring complex and expensive computations.

1. INTRODUCTION

To date, video delivery services over IP networks have achieved significant, but limited, success. In particular, as reported by several studies, real-time video transmission requires large and uninterrupted bandwidth, and rigid bounds on packet loss and delay to provide an acceptable quality of service. Because of the lack of such guarantees in the best-effort services, traditional approaches struggle to deliver the video packets to the client under strict delay requirements.

Many empirical studies (*e.g.*, [1, 2, 3]) report that end-users experience a considerable amount of delay jitter. While the amount of the reported jitter is the largest for dial-up modem users, cable customers usually observe more jitter compared to DSL subscribers. Evidently, jitter causes some packets to be late for decoding at the client. These excessively-delayed packets are essentially as useless as lost packets. Although it is possible to alleviate these adverse effects by increasing the playout delay in one-way streaming applications, this increase cannot be arbitrarily large. Preferably, the playout delay should be small enough so that client

requests (*e.g.*, forward and rewind operations) can be accommodated without incurring large re-buffering delays. Moreover, in two-way conversational applications, delay tolerance is severely limited in order to maintain the interactivity.

In addition to delaying packets, jitter has an implicit, but equally devastating, impact on the streaming quality. As the jitter aggravates, the time window for a decision gets narrower, and hence, it becomes more difficult to exactly know if a missing packet is actually lost or just delayed. Without a doubt, accurate and timely inference of late/lost packets will allow the streaming application to take well-timed actions for error control/protection. For example, in case of retransmission-based error-control methods, a retransmission at an early stage can be redundant as it might generate duplicate packets, whereas a late attempt will probably be useless. Hence, a precise retransmission timeout (RTO) estimation algorithm is extremely important. Naturally, as the delay tolerance of the application increases or the delay variation decreases, the client likely gives more accurate and well-timed retransmission decisions, which probabilistically increase the chance of timely delivery of the packets.

Precise RTO estimation algorithm can also be used in applications employing different types of error-control/protection methods. For example, one can adjust the level of redundancy in channel coding according to the up-to-date number of lost packets. For the applications where the streamed media is captured and encoded on the fly, the encoding process can utilize the list of late/lost packets and adapt itself accordingly such that the impact of the late/lost packets on the streaming quality is minimal. Clearly, any error-control method can exploit the potential benefits of accurate RTO estimation. Note that some of the error-control methods are not necessarily retransmission-based. However, following the TCP jargon, we will refer to our algorithm as retransmission timeout estimation.

In the past, several RTO estimation techniques have been evaluated for TCP (*e.g.*, [4, 5]). These studies considered conservative estimators since TCP packets did not have strict delivery deadlines and the main goal was to minimize the number of duplicate packets in the interest of being network-friendly. In its earlier versions, TCP used coarse-grained timers in estimating the mean and deviation of round-trip time (RTT). Since these estimations were biased, the resulting RTO values were not very accurate. Recently, there have been proposals to address this problem [6, 7]. Improving the granularity of the timers and using more effective filtering techniques proved to be performing better than the conventional algorithm proposed in [5]. However, these improved RTO estimators

are specifically designed for TCP, and hence, are not well suited to delay-sensitive applications.

For real-time streaming applications the most comprehensive work is [8], where Loguinov and Radha studied various RTO estimators based on the data collected from low-bitrate video streaming experiments between a video server and dial-up modem users. The trade-offs between the number of duplicate packets and the amount of unnecessary timeout waiting for these estimators were also investigated. Although [8] presents important findings on RTO estimation, it has two main shortcomings. First, the RTO estimation is based on the RTT samples. However, our experiments indicate that RTT samples potentially show a large variation because of the jitter experienced on both forward and backward paths. Furthermore, when RTT is sampled periodically, the correlation between the samples reduces, which subsequently renders RTO estimation difficult and degrades its performance. In contrast, by tracking the packet interarrival times measured at each packet arrival, one can estimate the arrival time of the next packet with a high accuracy since the variation of the packet interarrival times is usually smaller. Second, the experimental setup in [8] is quite different from ours, *e.g.*, while the authors experiment low-bitrate video streaming on modem users and keep the startup delay in the order of several seconds, we consider higher-bitrate video and limit playout delay to less than a second. This naturally leaves us with a very short time for decision. Hence, our RTO estimator should adapt to the network conditions more promptly and be aggressive as much as possible.

In this study, we introduce a new client-driven RTO estimation algorithm that can be used to improve the performance of any error-control method in real-time audio/video streaming applications. The main goal of our RTO estimator is to identify the lost packets as quickly as possible, while keeping the chance of any misidentification, *i.e.*, identifying a late packet as a lost packet, low. First, by studying the dynamics of packet delays, we develop an RTO estimator based on the packet arrival observations at the client side. Second, we apply our algorithm to a simple retransmission-based error-control method to experiment with an Internet video streaming application. The results clearly show that our approach can substantially improve the video quality delivered to the clients without requiring complex and expensive computations.

The rest of the paper continues with an overview of the simulations and experiments, and an introduction of the terminology. We discuss the development of the RTO estimator proposed in this work in Section 3. Section 4 presents the results produced from the Internet video streaming experiments. Finally, in Section 5 we conclude the paper with a discussion of future work.

2. METHODOLOGY

2.1. Simulations and Experiments

We conduct our analysis of packet delays with simulations because of two reasons. First, the analysis is easier when the network topology and traffic information are known a priori. Second, simulations allow us to experiment with various network topologies and characteristics, and hence, test the robustness of our RTO estimator. To this effect, we utilized *ns-2* network simulator [9]. We generated a moderate-sized Internet topology [10], where the nodes in each stub domain were associated with either a data traffic (running over TCP) or a video traffic (running over UDP at different

bitrates). Because of the uneven link delays and bandwidths, the resulting background traffic on each link gave us the opportunity to collect packet traces with different characteristics. We benefited from these packet traces in developing and fine-tuning the RTO estimator as discussed in Section 3.

After completing the simulations, we established an experimental platform in the Internet in order to assess the performance of our algorithm when used on real Internet traffic. 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.

During the experiments, the video streaming session lasted for 30 minutes. We used a standard H.264 codec to encode the test sequence FOREMAN (176×144) at 300 Kbps with a frame rate of 25 f/s. With these settings, we were able to pack each frame within a single video packet. Consequently, the server transmitted the video packets sequentially at every 40 ms. On the other hand, any retransmission request was answered immediately by the server.

As discussed previously, it is common to pre-buffer some initial video frames before starting to playout the stream in order to compensate for one-way delay jitter and produce some time to allow retransmissions. In our experiments, we employed a playout delay of 500 ms based on the observations. The packets that still could not be delivered by their decoding deadlines were not displayed, although late packets were still used to decode subsequent predictively-coded frames.

2.2. Terminology

Similar to [11], our RTO estimation approach is based on monitoring the packet arrivals at the client. Before giving the details, let us first introduce the terminology and notation that we will use throughout the paper. We associate three properties with each packet. For packet l , we denote its transmission time (at the server), arrival time (at the client) and decoding deadline by $t_T(l)$, $t_A(l)$ and $t_D(l)$, respectively. A packet is counted as successful if it is delivered to the client before its decoding deadline. Note that a successful packet does not necessarily mean a decodable packet because of some sort of possible interdependency relations among the packets. For the time being, we ignore this issue and defer its discussion until Section 4.

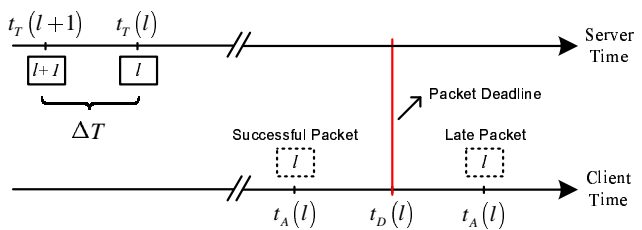


Fig. 1. Illustration of terminology.

In Fig. 1, ΔT represents the interval between the transmissions of consecutive packets. As mentioned above, ΔT equals to 40 ms in our experiments. However, in some cases video frames may produce a different number of packets depending on the video bitrate, encoding structure and packet size. If all the packets belonging to a frame are transmitted back-to-back in order to avoid any unnecessary waiting, the video traffic becomes bursty. Clearly,

the value of ΔT is no longer constant in such cases. We will investigate mechanisms that can handle this bursty behavior and its implications on RTO estimation in our future work.

Recall that $t_A(l)$ denotes the arrival time for packet l . Hence, we define interarrival time for packet l as

$$\Delta t(l) = t_A(l) - t_A(l^*), \quad (1)$$

where l^* is the index of the last received packet before packet l . For the non-received packets, we clearly have $t_A(l) = \infty$ and $\Delta t(l) = \infty$. It is important to note that in (1) l is an incremental identifier that is assigned to the packets at the time of transmission.

The fundamental principle of our RTO estimation algorithm is to predict the next arrival time after each packet arrival. Essentially, the algorithm estimates an interarrival time for the next expected packet. We will use the notation of $\widetilde{\Delta t}(\cdot)$ to denote the estimated interarrival times. For example, in Fig. 2 packet $l + 1$ is expected to arrive in $\widetilde{\Delta t}(l + 1)$ time units after the previous packet's arrival, $t_A(l)$. If the 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.

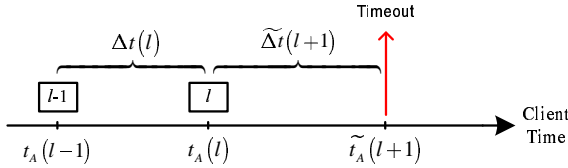


Fig. 2. RTO mechanism.

In evaluating the performance of an RTO estimator, there are two important criteria. The first one is the rate of misidentifying the late packets as they are lost. In other words, it is the frequency of underestimating the RTO values. Let $\widetilde{t}_A(l + 1)$ be the estimated arrival time for packet $l + 1$. It follows that

$$\widetilde{t}_A(l + 1) = t_A(l) + \widetilde{\Delta t}(l + 1). \quad (2)$$

Consequently, one can compute the rate of RTO underestimation as the following probability:

$$p_f = P\{\widetilde{t}_A < t_A\} \quad (3)$$

The second criterion is the average excessive waiting time. We define the excessive waiting time for a non-lost packet l as

$$w(l) = \begin{cases} \widetilde{t}_A(l) - t_A(l), & \text{if } t_A(l) < \widetilde{t}_A(l) < \infty; \\ 0, & \text{if } \widetilde{t}_A(l) \leq t_A(l) < \infty. \end{cases} \quad (4)$$

On the other hand, in order to account for the waiting time spent for a lost packet, we use a virtual arrival time that is extrapolated from the latest packet arrival time. If l^* denotes the index of the last received packet, for a lost packet its excessive waiting time is defined as

$$w(l) = \widetilde{t}_A(l) - (t_A(l^*) + (l - l^*) \times \Delta T), \text{ if } t_A(l) = \infty. \quad (5)$$

With (4) and (5), we award the early detection of lost packets and penalize the excessive waiting, which is very critical for delay-sensitive applications. As a rule of thumb, RTO estimators with smaller average excessive waiting times should be preferred as long as the rate of RTO underestimation is kept below a certain value.

3. RTO ESTIMATION

Let us start our discussion with the analysis of two packet traces that are obtained at 300 and 600 Kbps. With an IP packet size of 1500 bytes, the corresponding initial gaps between the consecutive packets are 40 and 20 ms, respectively. For the sake of clarity, in Fig. 3 - 6 we present the forward-trip times and interarrival times extracted from a small segment of the traces. The distributions are also given in Fig. 7 and 8, respectively. Recall that for the lost packets, arrival times and interarrival times are equal to ∞ and are not shown.

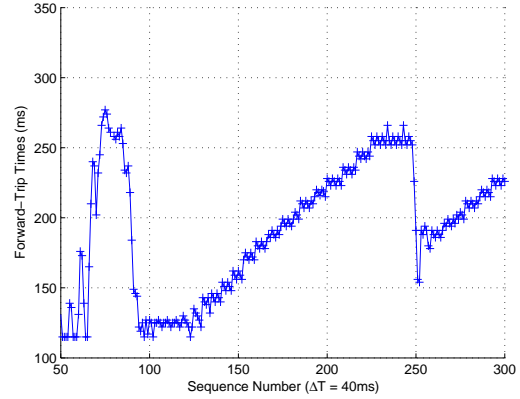


Fig. 3. Forward-trip times for $\Delta T = 40$ ms.

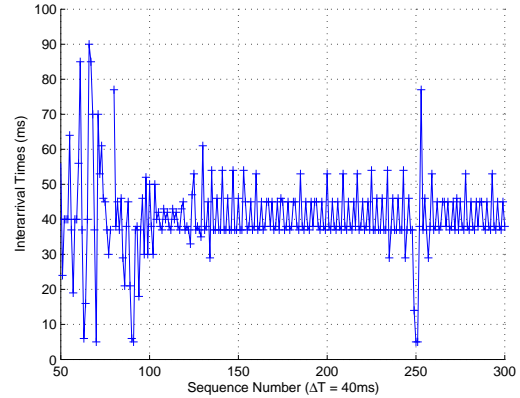


Fig. 4. Interarrival times for $\Delta T = 40$ ms.

Although the particular values presented here are dependent on the settings and topology adopted during the simulations, there are some features common to almost all packet traces. A first look on the distributions shows that forward-trip times have larger variations compared to the interarrival times. A further examination of Fig. 4 and 6 reveals that interarrival times are mostly confined within a small region around ΔT (except a few impulsive points). This makes the interarrival times easier to predict. It is interesting that although the distribution of forward-trip times seems not to vary a lot with ΔT , the variation of the interarrival times reduces as ΔT gets smaller.

Before presenting the details of our RTO estimation algorithm, we briefly summarize the RTO estimation in TCP. We will later use an enhanced version of this estimator as a benchmark in evaluating the performance of the proposed estimator.

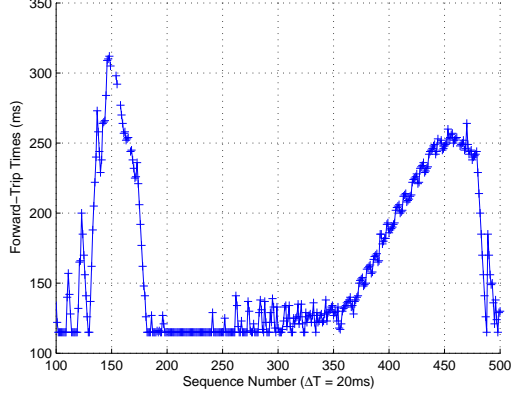


Fig. 5. Forward-trip times for $\Delta T = 20$ ms.

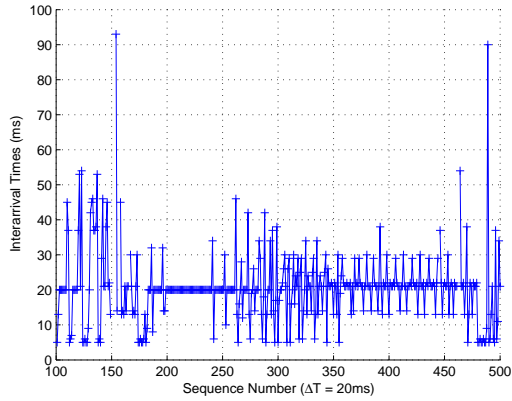


Fig. 6. Interarrival times for $\Delta T = 20$ ms.

3.1. TCP-Like RTO Estimators

The RTO estimation algorithm used in current TCP implementations is based on Jacobson's algorithm [5], which has been later modified in [12]. When a TCP sender measures a new round-trip time (RTT_{sample}), the smoothed RTT ($SRTT$) estimate is updated by using a low-pass filter with a weighing factor of $7/8$ as follows:

$$SRTT = \frac{7}{8} \times SRTT + \frac{1}{8} \times RTT_{sample}. \quad (6)$$

The TCP sender also keeps track of the variation in RTT_{sample} , which is computed by

$$\sigma_{RTT} = \frac{3}{4} \sigma_{RTT} + \frac{1}{4} \times |RTT_{sample} - SRTT|. \quad (7)$$

Subsequently, the value of RTO is computed by using

$$RTO = \max(RTO_{min}, SRTT + \max(G, 4 \times \sigma_{RTT})), \quad (8)$$

where G is the clock granularity and RTO_{min} is used to avoid spurious timeouts. The suggested value for RTO_{min} is one second [12]. Note that current TCP variants also implement Karn's algorithm [13], which ignores the RTT measurements for the retransmitted packets to circumvent a possible ambiguity. This algorithm also suggests to double the RTO value when a timeout occurs. While exponential timer backoff is practical for TCP flows, it is usually too costly for delay-sensitive applications.

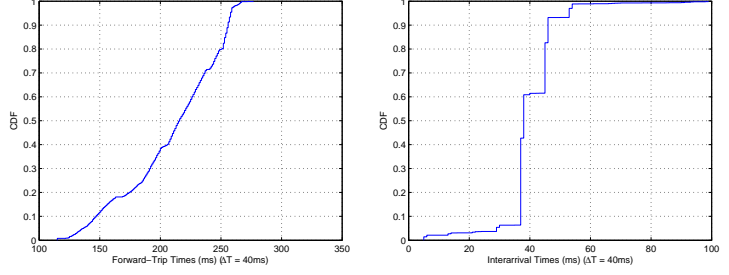


Fig. 7. Distributions of forward-trip and interarrival times for $\Delta T = 40$ ms.

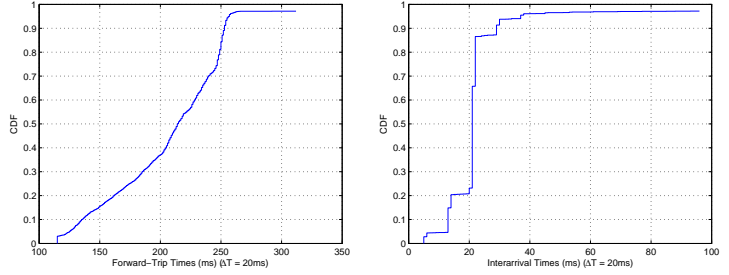


Fig. 8. Distributions of forward-trip and interarrival times for $\Delta T = 20$ ms.

If we consider that the traffic on the forward and backward paths are loosely correlated, then it is clear that the round-trip times will inevitably suffer from the variations in both forward and backward-trip times. Consequently, the value of σ_{RTT} will be inflated resulting in overvalued RTO values. In order to remove this adverse effect, we will adopt (8) to estimate the forward-trip times during our comparisons. Furthermore, we will set RTO_{min} to zero and use a clock granularity of 10 ms to reduce excessive waiting. This estimator will be referred to as enhanced TCP-like RTO estimator.

3.2. The Proposed Approach

Generally, continuous-media applications inject less-bursty traffic into the network compared to TCP flows. As a result, the clients receive packets more regularly. This feature enables them to monitor the packet arrivals and infer the status of the missing packets. In contrast to its server-based counterpart, the information at the client side is expected to be more accurate. Hence, we propose a solely client-driven RTO estimation algorithm.

As mentioned in Section 2, the main idea is to predict the next arrival time after each packet arrival. Our prediction is based on the latest interarrival time. Specifically, we use

$$\tilde{\Delta t}(l+1) = \max(\Delta T, \alpha \times \Delta t(l) + \beta \times \Delta T), \quad (9)$$

where $\beta \leq \alpha \leq 1$. With $\alpha = 0.875$ and $\beta = 0.375$ (found by trial and error), this aggressive estimator inevitably causes spurious timeouts, although it achieves very low average excessive waiting time. For the 300 Kbps (600 Kbps) simulations, the observed rate of RTO underestimation is 9.8% (19.3%) and the average excessive waiting time is 12 ms (8 ms). On the other hand, the enhanced TCP-like estimator in (8) causes spurious timeouts

with a probability of 1.1% (0.7%), while resulting in an average excessive waiting time of 84 ms (119 ms). When the actual and estimated arrival times are examined, it is easy to see that the impulsive points in the interarrival times are responsible for the high failure rate of the proposed RTO estimator. We address this problem next.

The estimator in (9) requires a packet arrival to compute the subsequent arrival time. Hence, in case of a bursty packet loss or an excessively-delayed packet, this estimator may halt. To avoid such interruptions and enable a continual operation, the interarrival time estimates are all initialized to ΔT . When a new packet arrives, the estimate for the subsequent packet is updated accordingly.

Because of the bursty behavior of TCP flows in the background, it is possible to observe some abrupt increases in the packet interarrival times (e.g., see packet #253 in Fig. 4 and packet #154 in Fig. 6). To avoid any misidentification, these impulsive points must be caught by the RTO estimator. Since these sudden increases occur rarely in the interarrival times, rather than modifying (9), e.g., increasing the value of β , we introduce a new concept, so-called *latePacketTimer*. This is basically a supplementary timer that tries to absorb large delays. The client starts *latePacketTimer* when the expected packet does not arrive by the estimated arrival time, as depicted in Fig. 9. If *latePacketTimer* also expires, then the client registers the packet as lost. When a new packet (either the expected one or another packet) is received, the client updates the duration of *latePacketTimer*, denoted by δ , and estimates the arrival time for the next packet. It is important to 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, this allows to keep the waiting times shorter.

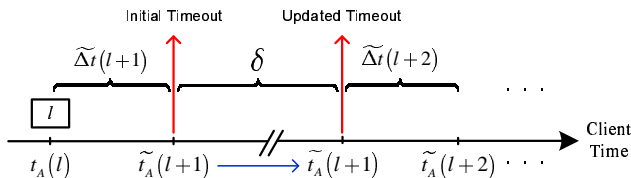


Fig. 9. Illustration of *latePacketTimer*.

With the introduction of *latePacketTimer*, the rate of RTO underestimation in the 300 Kbps (600 Kbps) simulations reduces to 0.25% (0.4%). The incurred cost is a small increase from 12 to 16 ms (8 to 12 ms) in the average excessive waiting time. The substantial gains at the expense of the negligible costs clearly prove the usefulness of employing *latePacketTimer* in the RTO estimation.

In the implementation, the initial value of δ can be set to zero since it is updated every time the timer is used. Let l_p and l_n be the indexes of the last received packet before *latePacketTimer* is started and the first packet received after *latePacketTimer* is started, respectively. Then, δ is set as follows:

$$\delta = \min(3 \times \Delta T, t_A(l_n) - t_A(l_p)) \quad (10)$$

Evidently, the enhanced TCP-like RTO estimator causes more spurious timeouts even though its average excessive waiting time is

longer. This is mainly because of the slow convergence of (8) and its susceptibility to delay variations. This can be easily seen in Fig. 10 and 11.

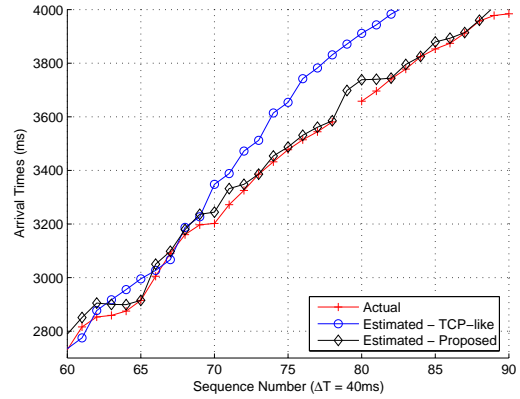


Fig. 10. Actual and estimated arrival times for $\Delta T = 40$ ms.

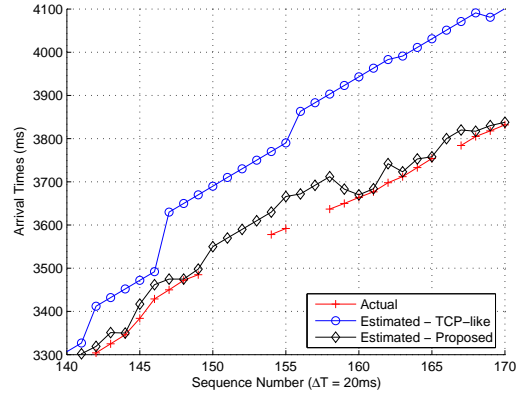


Fig. 11. Actual and estimated arrival times for $\Delta T = 20$ ms.

4. EXPERIMENTAL RESULTS

The goal of the RTO estimator we proposed in Section 3 was to identify the missing packets as quickly as possible. In this section, we apply this estimator to a retransmission-based error-control method for experimenting with a real-time video streaming application in the Internet. The client simultaneously streams real-time video and carries out the RTO computation. When a packet is identified to be lost, a retransmission request is sent to the server provided that there is sufficient time for a retransmission. If this request is received successfully, the server immediately replies back with the requested packet. Further details of the experimental setup are discussed in Section 2.

In real-time streaming, timely reception of a packet does not guarantee its decoding. This is because of the highly-efficient predictive-encoding techniques that are used in popular video coding standards, e.g., MPEGx and H.26x. As a result of predictive coding, a dependency is created among the video packets that determines the order in which the packets will be decoded. For example, to be able to successfully decode and display a packet (e.g.,

a P-frame), the client has to receive and decode all the corresponding *ancestor* packets (e.g., the I-frame and previous P-frames in the same GOP) by the packet's decoding deadline. Likewise, a lost/late packet not only causes quality degradation during its display, but also impedes the successful decoding of its *descendant* packets.

Without a doubt, the dependency structure of video renders video packets unequally important. However, our goal in this section is to demonstrate the superior performance of the proposed RTO estimator rather than achieving the highest quality of video. To this effect, we consider each video packet individually and merely deal with the timeliness requirements. In our future work, we will investigate media-aware RTO estimation algorithms that will consider the interdependency structure among the video packets in addition to their decoding deadlines.

We conducted our experiments in two sessions of 30 minutes, to evaluate the two RTO estimators. After each session, we measured the mean delay and packet loss rate to ensure that similar network characteristics were observed in both sessions. (The mean round-trip delay and mean packet loss rate were approximately 250 ms and 5.8%, respectively.) In the experiments, every packet was allowed for a maximum of one retransmission. We summarize our results in Table 1.

Table 1. Experimental results for the FOREMAN sequence.

	Success Rate	Avg. PSNR	p_f	\bar{w}
TCP-like	95.4%	35.8 dB	1.2%	125 ms
Proposed	98.5%	38.6 dB	0.6%	23 ms

The results demonstrate that the proposed RTO estimator delivers 3.1% more packets on time, resulting in an average PSNR improvement of 2.8 dB. At the same time, its failure probability and the incurred average excessive waiting time are smaller than those of the enhanced TCP-like estimator.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an RTO estimation algorithm for delay-sensitive applications. In contrast to current implementations, this algorithm solely works at the client side and utilizes packet inter-arrival times in the estimation. By Internet experiments, we show substantial improvements in the timely inference of late/lost packets. Specifically, compared to an enhanced TCP-like RTO estimator, the average excessive waiting time is reduced by 80-90% and the probability of misidentifying the late packets as lost is reduced by 42-77%. When a simple retransmission-based error-control method is employed with the proposed algorithm, these reductions translate to a 2.8 dB improvement in the video quality.

An important aspect of the proposed RTO mechanism is that it can be integrated with the recently proposed rate-distortion optimization techniques. Motivated by the pioneering work of Chou and Miao [14], several authors addressed rate-distortion optimization in different streaming scenarios. Essentially, the common goal is to find the optimal transmission policies for the packetized media by utilizing a Markov decision process (MDP) framework. However, in order to simplify the computations this framework assumes independent packet loss events and packet delays. While these assumptions can be justified in low-bandwidth streaming applications, our experiments and several other studies (e.g., [15, 16,

2]) show that they are often insufficient to model the actual packet dynamics for high-bandwidth streaming applications.

As a matter of fact, introducing the Markovian models [15, 16] for packet loss into the MDP framework increases its complexity and requires more computationally-intensive calculations. Fortunately, the methodology presented in this paper can be incorporated into any of the previously proposed rate-distortion optimized streaming methods, regardless of the media type and coding scheme. Not only it enables the packet scheduling algorithms to give more precise decisions, but also it obviates the complex calculations. As a result, one can achieve a more *reliable* and *realizable* real-time rate-distortion optimized streaming. In our future work, we will study the integration of our RTO estimation algorithm with the rate-distortion optimized streaming framework.

Acknowledgments

This work is supported by NSF under NSF CAREER award CCR-0133221.

6. REFERENCES

- [1] Y. Wang, M. Claypool, and Z. Zuo, "An empirical study of realvideo performance across the internet," in *ACM SIGCOMM Internet Measurement Workshop*, 2001.
- [2] D. Loguinov and H. Radha, "End-to-end internet video traffic dynamics: Statistical study and analysis," in *IEEE Int. Conf. Computer Communications (INFOCOM)*, 2002.
- [3] K. Lakshminarayanan and V. Padmanabhan, "Some findings on the network performance of broadband hosts," in *ACM Internet Measurement Conf.*, 2003.
- [4] M. Allman and V. Paxson, "On estimating end-to-end network parameters," in *ACM SIGCOMM*, 1999.
- [5] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM*, 1988.
- [6] 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.
- [7] R. Ludwig and K. Sklower, "The eifel retransmission timer," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 3, pp. 17–27, 2000.
- [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] S. McCanne and S. Floyd. Network simulator. [Online]. Available: <http://www.isi.edu/nsnam/ns>
- [10] 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.
- [11] R. Gupta, M. Chen, S. McCanne, and J. Walrand, "A receiver-driven transport protocol for the web," *Telecommunication Systems Journal*, vol. 21, no. 2-4, pp. 213–230, December 2002.
- [12] Computing TCP's Retransmission Timer. [Online]. Available: <http://www.ietf.org/rfc/rfc2988.txt>
- [13] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," in *ACM SIGCOMM*, 1987.
- [14] P. A. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *Microsoft Research Technical Report MSR-TR-2001-35*, 2001.
- [15] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, "Measurement and modelling of the temporal dependence in packet loss," in *IEEE Int. Conf. Computer Communications (INFOCOM)*, 1999.
- [16] J. Wenyu and H. Schulzrinne, "Modeling of packet loss and delay and their effects on real-time multimedia service quality," in *ACM NOSSDAV*, 2000.