EFFECTS OF CHANNEL DELAYS ON UNDERFLOW EVENTS OF COMPRESSED VIDEO OVER THE INTERNET

Dmitri Loguinov
City University of New York
New York, NY 10016
csdsl@cs.ccny.cuny.edu

Hayder Radha
Michigan State University
East Lansing, MI 48824
radha@egr.msu.edu

ABSTRACT

This paper presents an extensive statistical study and analysis of the effects of channel delays in the current (best-effort) Internet on underflow events in MPEG-4 video streaming. Two types of network delays are considered: end-to-end round-trip delays and delay jitter. Our data were collected in a seven-month real-time streaming experiment, which was conducted between a number of unicast dialup clients in more than 600 major U.S. cities and a backbone video server. Among other findings, our analysis shows that startup delays approximately 15-20 times the average round-trip time (RTT) are required for the client to avoid 90% of late packets caused by delay jitter. Meanwhile, startup delays of only 3-4 times the average RTT are needed to achieve lost-packet recovery rates of 90% or more. Hence, a key finding of our study is that delay jitter represents a more challenging problem for video streaming applications than round-trip delays. We also show that the probability density function (PDF) of RTT samples, which are associated with retransmitted video packets, can be modeled using a Pareto distribution. This observation indicates that the upper tail of the RTT PDF decays slower than reported in earlier studies.

1. INTRODUCTION

Real-time streaming in the current Internet is dominated by two popular applications – Real Player [7] from Real Networks and Windows Media Player [12] from Microsoft. These applications typically require that the user explicitly select the desired streaming rate from the corresponding web page and are frequently used by end-users for *constant bitrate* (CBR) streaming. The exact streaming protocols and system-level bitstreams used in both applications [7], [12] remain proprietary, which explains why only a few research studies focused on this type of Internet traffic (e.g., [4], [11]). Furthermore, the absence of prior *active* measurement studies (as opposed to *passive* studies [4]) of real-time streaming in the Internet is another reason that prompted us to conduct a large-scale real-time streaming experiment to study the end-to-end dynamics of video streaming in the best-effort Internet, investigate the effect of large end-to-end delays on the video quality (i.e., the frequency of underflow events), and analyze the overall effectiveness of real-time retransmission.

Even though the end-to-end performance of the Internet has been extensively analyzed in the past, an overwhelming majority of previous studies were based on TCP or ICMP traffic. Note that current streaming applications [7] often rely on NACK-based flow control, which makes them quite different from TCP and other ACK-based transport protocols. On the other hand, real-time streaming protocols have not received as much attention in these studies. In fact, the dynamics of NACK-based UDP protocols (not necessarily real-time) have been the focus of only a few research studies (e.g., [3]).

Internet performance parameters have been studied using TCP traffic [6], ICMP packets [6], UDP echo packets [2] and multicast backbone (MBone) audio packets [10]. With the exception of the last study, neither the setup, nor the type of probe traffic of prior work represented realistic real-time streaming scenarios. In addition, among the studies that specifically sent multimedia traffic over the Internet (e.g., [1], [9]), the majority of experiments involved only a few Internet paths, lasted for a short period of time, and focused on analyzing the features of the proposed scheme rather than the impact of Internet conditions on real-time streaming.

The methodology used in previous large-scale TCP experiments (e.g., [6]) was similar and involved a topology where each participating site was paired with every other participating site for an FTP-like transfer. Although this setup approximates well the current use of TCP in the Internet, future entertainment-oriented streaming services, however, are more likely to involve a small number of backbone video servers and a large number of home users.[1]

Furthermore, in order to study the current dynamics of real-time streaming in the Internet, it is crucial to take the same steps to connect to the Internet as an *average end-user*[2] (i.e., through dialup ISPs). For example, ISPs often experience congestion in their own backbones, and during busy hours, V.90 modems in certain access points are not available due to high user demand, none of which can be captured by studying the Internet from a small campus network directly connected to the Internet backbone. Consequently, our topological setup involved a single backbone server and a large number of unicast dialup clients connecting to the Internet from all parts of the U.S. (more on this in the next section).

The remainder of the paper is organized as follows. Section 2 describes the methodology of the experiment and provides an overview of the experiment. Section 3 analyzes the end-to-end delay jitter and round-trip delay and their impact on video underflow events. We also discuss the effectiveness of real-time retransmission for packet loss recovery in the context of our findings. Section 4 concludes the paper.

2. METHODOLOGY

2.1. Setup of the Experiment

As mentioned above, in order to provide a realistic and robust evaluation of the end-to-end delays present in streaming applications, it is crucial to employ an experimental setup that emulates a typical end-user access to the Internet. Our experiment topology consisted of a Unix video server (located in the state of New York,

[1] Our work focuses on non-interactive streaming applications where the user can tolerate short (i.e., in the order of several seconds) startup delays (e.g., TV over the Internet).

[2] Recent market research reports (e.g., [7]) show that in Q2 of 2001, approximately 87% of U.S. households used dialup access to connect to the Internet. Furthermore, it is predicted [7] that even in 2005, the majority of U.S. households will still be using dialup modems.

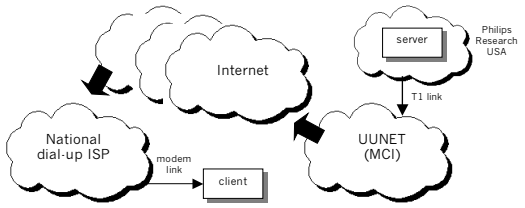

Figure 1. Topology of the experiment.

USA, see Figure 1) and a large number of diverse dialup clients. To support the clients' connectivity to the Internet, we selected three major nation-wide dialup ISPs (which we call ISP_a, ISP_b, and ISP_c), each with at least five hundred V.90 (i.e., 56 kb/s) access numbers in the U.S., and designed an experiment in which hypothetical Internet users (residing in all 50 states) dialed a local access number to reach the Internet and streamed video sequences from the server. Although the clients were physically placed in our New York lab, they dialed long-distance phone numbers and connected to the Internet through ISPs' access points located in each of the 50 states. Our database of phone numbers included 1,813 different V.90 access points in 1,188 major U.S. cities.

We implemented an automated program (*dialer*) that (1) dialed phone numbers from the database mentioned above, (2) connected to the ISPs using the point-to-point protocol (PPP), and (3) issued a traceroute to the server. Upon success of the traceroute, the dialer started the video client with the instructions to stream a ten-minute video sequence from the server.

In our analysis of the data, we attempted to isolate clearly modem-related pathologies (such as packet loss caused by a poor connection over the modem link and large RTTs due to data-link retransmission) from those caused by congested routers of the Internet. Thus, connections that were unable to complete a traceroute to the server, connections with high bit-error rates[3] (BER), and connections during which the *modem* could not sustain our streaming rates were all excluded from the analysis in this paper.

2.2. Real-time Streaming

For the purpose of the experiment, we used an MPEG-4 encoder to create two ten-minute QCIF video streams. The first stream, which we call S_1, was coded at the video bitrate of 14 kb/s (size 1.05 MBytes), and the second steam, which we call S_2, was coded at 25 kb/s (size 1.87 MBytes). The combined experiment with streams S_1 and S_2 lasted for seven months in 1999-2000.

During the transmission of each video stream, the server split it into 576-byte IP packets. Stream S_1 consisted of 4,188 packets, and stream S_2 consisted of 5,016 packets. Video frames always started on a packet boundary, and the last packet in each frame was allowed to be smaller than others (in fact, many P frames were smaller than the maximum payload size and were carried in a single UDP packet). As a consequence of packetization overhead, the *IP bitrates* (i.e., including IP, UDP, and our 8-byte headers) for streams S_1 and S_2 were 16.0 and 27.4 kb/s, respectively.

In our streaming experiment, the term *real-time* refers to the fact that the video decoder was running in real-time. Recall that each compressed video frame has a specific *decoding deadline*, which is usually based on the time of the frame's encoding. If a compressed video frame is not fully received by the decoder buffer at the time of its deadline, the video frame is discarded and an underflow event is registered. In order to compensate for one-way delay jitter and allow retransmissions, many real-time streaming protocols *pre-buffer* video data before starting the decoding process [7], [12]. The duration of such pre-buffering is called the *startup delay* of a session. In all our experiments, we used a startup delay equal to 2,700 ms.

2.3. Client-Server Architecture

For the purpose of our experiment, we implemented a NACK-based client-server architecture for MPEG-4 streaming over the Internet. Our client was designed to recover lost packets through NACK-based retransmission and collect extensive statistics about each received packet and each decoded frame. Furthermore, the client was in charge of collecting round-trip delay (RTT) samples. The measurement of the RTT involved the following two methods. In the first method, each successfully recovered packet provided a sample of the RTT (i.e., the RTT was the duration between sending a NACK and receiving the corresponding retransmission). The second method of measuring the RTT was used by the client to obtain *additional* samples of the round-trip delay in cases when network packet loss was too low. The method involved periodically sending *simulated* retransmission requests to the server if packet loss was below a certain threshold. In our experiment, the client activated simulated NACKs, spaced 30 seconds apart, if packet loss was below 1%.

We tested the software, our retransmission algorithm, and the concept of a wide-scale experiment of this sort for nine months before we felt comfortable with the setup, the reliability of the software, and the exhaustiveness of the collected statistics. Our traces consist of six datasets, each collected by a different machine. Throughout this paper, however, we will use notation D_n to refer to the combined dataset collected during the experiment with stream S_n (n = 1, 2) using the dialup points of all three ISPs.

Key statistical data regarding the dynamics of our real-time Internet experiment are summarized in Appendix A. These data illustrate the extensiveness of coverage of Internet paths (over 5,000 different Internet routers traced from 653 U.S. cities) and show that our results are based on transmission dynamics of over 85 million packets, which makes these results statistically valid.

3. IMPACT OF NETWORK DELAYS ON VIDEO BUFFER UNDERFLOW EVENTS

The impact of packet losses on real-time applications is understood fairly well. Each lost packet that is not recovered before its deadline causes an underflow event. In addition to packet loss, real-time applications suffer from large end-to-end delays. However, not all types of delay are equally important to real-time applications. As we will show below, one-way delay jitter was responsible for 90 times more underflow events in our experiment than packet loss combined with large RTTs.

Delays are important for two reasons. First, large round-trip delays make retransmissions late for their decoding deadlines. However, the RTT is important only to the extent of recovering lost packets and, in the worst case, can cause only *lost* packets to be late for decoding. On the other hand, the second kind of delay, delay jitter (i.e., one-way delay variation), can potentially cause each *data* (i.e., non-retransmitted) packet to be late for decoding.

3.1. Round-trip Delays and Retransmission

Before presenting the roundtrip delay data, it is important to highlight that one of the key findings of our study is that a high majority of our unicast streaming sessions experienced packet loss

[3] Since the telephone network beyond the local loop in the U.S. is mostly digital, we believe that dialing long-distance (rather than local) numbers had no significant effect on the number of bit errors during the experiment. Furthermore, the additional propagation delay of the long-distance circuit was negligibly small to have a noticeable impact on the end-to-end delays.

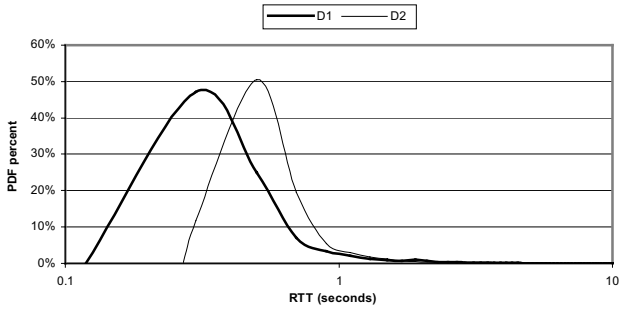

Figure 2. PDFs of the round-trip delay.

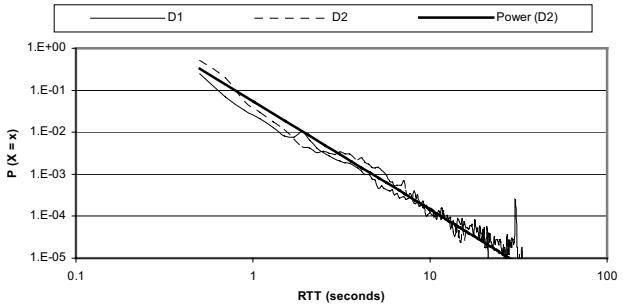

Figure 3. Upper tails of the RTT PDFs on a log-log scale.

rates of less than 1%. This indicates that high packet-loss combined with large round-trip delays may not be as serious an issue for streaming applications as has been previously feared. Nevertheless, even these relatively low packet loss ratios could cause major degradation to the overall video quality. Therefore, one key question we attempted to answer in our study is: how much buffering delay should be used to provide on-time recovery of lost packets through retransmission? We show below that: (1) relatively small buffering delays are quite adequate to recover a vast majority (i.e., over 90%) of lost packets through retransmission; (2) the RTT PDF tail tends to decay slowly, and this indicates a virtual impossibility of recovering all lost packet (i.e., 100% recovery) regardless of the amount of buffering delay used.

Consider the following. In $\{D_1 \cup D_2\}$, packet loss affected 431,501 packets, out of which 159,713 (37%) were discovered to be missing *after* their decoding deadlines had passed, and consequently, NACKs were not sent for these packets. Out of 271,788 remaining lost packets, 257,065 (94.6%) were recovered before their deadlines, 9,013 (3.3%) arrived late, and 5,710 (2.1%) were never recovered. The fact that more than 94% of "recoverable" lost packets were actually received before their deadlines indicates that retransmission is an efficient method of overcoming packet loss in real-time applications. Clearly, the recovery success rate will be even higher in networks with smaller end-to-end delays, but it will also depend on the amount of pre-buffering applied to the decoder buffer and its relationship to the average RTT.

Out of 660,439 RTT samples in the combined dataset (Figure 2), the majority (75%) were below 600 ms, 90% below 1 second, and 99.5% below 10 seconds. The average RTT was 698 ms in D_1 and 839 ms in D_2. The minimum RTT was 119 and 172 ms, respectively. These numbers suggest that a startup delay approximately four times higher than the average RTT is sufficient to recover over 90% of "recoverable" packets in the dialup Internet. At the end of this section, we will examine in more detail the tradeoff between the amount of startup delay and the success rate of real-time retransmission during our experiment.

Mukherjee [5] reported that the distribution of the RTT along certain Internet paths could be modeled as a shifted gamma distribution. Even though the shape of the PDF functions in Figure 2 resembles that of a gamma function, the distribution tails in the figure decay much slower than those of an exponential distribution. To investigate this further, we extracted the upper tails of the PDF functions in Figure 2 and plotted them on a log-log scale in Figure 3. The figure shows that a straight line (without loss of generality fitted to the PDF of D_2) provides a good fit to the data (correlation 0.96) and lets us model the upper tails of the PDF functions in Figure 3 as a hyperbolic Pareto distribution with CDF $F(x) = 1-(\beta/x)^{\alpha}$ and PDF $f(x) = \alpha\beta^{\alpha}x^{-\alpha-1}$, where shape parameter α equals 1.16 in dataset D_1 and 1.58 in D_2.

Before we study underflow events caused by delay jitter, let us introduce two types of *late* retransmissions. The first type consists of packets that arrived after the decoding deadline of the last frame of the corresponding group of pictures (GOP). These packets were *completely* useless and were discarded. The second type of late packets, which we call *partially late*, consists of those packets that missed their *own* decoding deadline, but arrived before the deadline of the last frame of the same GOP. Since the video decoder in our experiment could decompress frames at a substantially higher rate than the target fps, the client was able to use partially late packets for motion-compensated reconstruction of the remaining frames from the same GOP before *their* corresponding decoding deadlines. Out of 9,013 late retransmissions, 4,042 (49%) were partially late. Using each partially late packet, the client was able to save on average 4.98 frames from the same GOP[4] in D_1 and 4.89 frames in D_2 by employing the above-described catch-up decoding technique. This result suggests that retransmitted packets often missed their deadlines by a small amount and frequently arrived while the same GOP was still playing.

3.2. Delay Jitter

The second type of delay, one-way delay jitter, caused 1,167,979 *data* (i.e., non-retransmitted) packets to miss their decoding deadlines. Hence, the total number of *underflow* (i.e., missing at the time of decoding) packets was 159,713 + 9,013 + 5,710 + 1,167,979 = 1,342,415 (1.7% of the number of sent packets), which means that 98.9% of underflow packets were created by large one-way delay jitter, rather than by pure packet loss. Even if the clients had not attempted to recover any lost packets, still 73% of the missing packets at the time of decoding would have been caused by large delay jitter. Furthermore, these 1.3 million underflow packets caused a "freeze-frame" effect for the average duration of 10.5 seconds per ten-minute session in D_1 and 8.6 seconds in D_2, which can be considered excellent given the small amount of delay budget (i.e., startup delay) used in the experiments.

We analyzed the distribution of *expansion* delay-jitter samples and found that 97.5% of them were under 140 ms and 99.9% under 1 second. Large values of delay jitter were not frequent, but once a packet was significantly delayed by the network, a substantial number of the following packets were delayed as well, creating a "snowball" of late packets. This fact explains the large number of underflow events reported above, even though the overall delay jitter was relatively low.

3.3. Startup Delay

To further understand the phenomenon of late packets, we plotted in Figure 4 the CDFs of the amount of time by which late packets missed their deadlines (i.e., the amount of time that we

[4] We used 10-frame "IPPPPPPPPP" GOPs in both sequences.

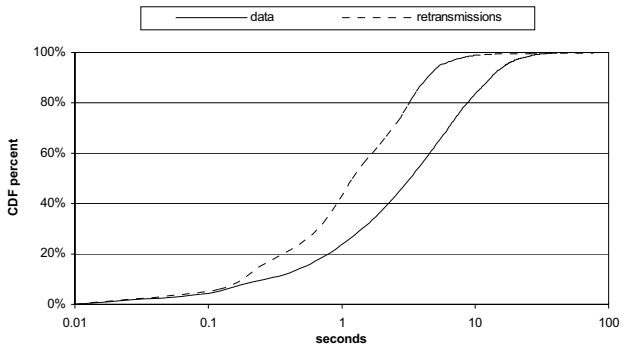

Figure 4. CDF functions of the amount of time by which retransmitted and data packets were late for decoding.

need to add to the startup delay used in the experiment in order to avoid a certain percentage of underflow events) for both late retransmissions and late data packets. As the figure shows, 25% of late retransmissions missed their deadlines by more than 2.6 seconds, 10% by more than 5 seconds, and 1% by more than 10 seconds (the tail of the CDF extends up to 98 seconds). At the same time, one-way delay jitter had a more adverse impact on data packets – 25% of late data packets missed their deadlines by more than 7 seconds, 10% by more than 13 seconds, and 1% by more than 27 seconds (the CDF tail extends up to 56 seconds).

The only way to reduce the number of late packets caused by both large RTTs and delay jitter is to apply a larger startup delay at the beginning of a session (in addition to freezing the display and adding extra startup delays during the session, which was not acceptable in our model). For example, Internet applications utilizing a 13-second startup delay (which corresponds to 10.3 seconds of *additional* delay in Figure 4 and equals 15 times the average RTT) would be able to "rescue" 99% of late retransmissions and 84% of late data packets in similar streaming conditions.

4. CONCLUSION

Our study found that large end-to-end delays did not pose much impediment to real-time retransmission in the dialup Internet. The majority (94%) of "recoverable" lost packets returned to the client before their decoding deadlines even in the presence of a relatively low startup delay of 2,700 ms. We also found that approximately 95% of all recovered packets were recovered using a single retransmission attempt (i.e., a single NACK). Nevertheless, we feel that the current end-to-end delays in the dialup Internet are prohibitively high to support interactive real-time applications (such as video conferencing or telephony). For non-interactive applications, startup delays in the order of 10-15 seconds (approximately 15-20 times the average RTT) are recommended given a random access point used in streaming; however, startup delays as low as one second were found to be acceptable over certain paths during the night. Even using forward error correction (FEC) coding instead of retransmission to overcome packet loss does not allow us to substantially lower the startup delay, because one-way delay jitter in the dialup Internet is often very large.

We speculate that end-to-end delays under 100 ms will be required to support interactive streaming, which seems to be currently possible with DSL and certain cable modems. Furthermore, we believe that with broadband access at home, the performance of real-time streaming will largely depend on the end-to-end congestion control employed in the streaming protocol, rather than on the backbone Internet packet-loss rates, a particular retransmission scheme, or delay jitter (all of which are significantly less relevant given low end-to-end delays).

5. REFERENCES

[1] J-C. Bolot and T. Turletti, "Experience with Rate Control Mechanisms for Packet Video in the Internet," *ACM Computer Communication Review*, January 1998.

[2] J-C. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," *ACM SIGCOMM*, September 1993.

[3] D. Loguinov and H. Radha, "On Retransmission Schemes for Real-time Streaming in the Internet," *IEEE INFOCOM*, April 2001.

[4] A. Mena and J. Heidemann, "An Empirical Study of Real Audio Traffic," *IEEE INFOCOM*, March 2000.

[5] A. Mukherjee, "On the Dynamics and Significance of Low-Frequency Components of Internet Load," *Internetworking: Research and Experience*, vol. 5, no. 4, December 1994.

[6] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," *Ph.D. dissertation*, UC Berkeley, 1997.

[7] S.P. Pizzo, "Why Is Broadband So Narrow?" *Forbes ASAP*, September 2001, p. 50.

[8] RealPlayer, Real Networks, http://www.real.com.

[9] W. Tan and A. Zakhor, "Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol," *IEEE Transactions on Multimedia*, vol. 1, no. 2, June 1999.

[10] M. Yajnik, S. Moon, J. Kurose, and D. Townsley, "Measurement and Modelling of the Temporal Dependence in Packet Loss," *IEEE INFOCOM*, March 1999.

[11] Y. Wang, M. Claypool, and Z. Zuo, "An Empirical Study of RealVideo Performance Across the Internet," *ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2001.

[12] Windows Media Player, http://windowsmedia.com.

Appendix A. Experiment Overview

In dataset D_1, the three clients performed 16,783 long-distance connections to the ISPs' remote modems and successfully completed 8,429 streaming sessions.[5] In D_2, the clients performed 17,465 modem connections and sustained 8,423 successful sessions. Analysis of the above numbers suggests that in order to receive real-time streaming material with a minimum quality at 16 to 27.4 kb/s, an average U.S. end-user, equipped with a V.90 modem, needs to make approximately two dialing attempts to the ISPs' phone numbers within the state where the user resides. Furthermore, the success rate of streaming attempts during different times of the day varied between 40% (9 am – noon EST) and 80% (3–6 am EST).

In the combined dataset $\{D_1 \cup D_2\}$, the clients traced the arrival of over 85 million packets, which accounted for 27 GBytes of video data, and sampled end-to-end Internet paths through 1,003 different access points in 653 major U.S. cities. Recall that during the experiment, each session was preceded by a traceroute, which recorded the IP addresses of all discovered routers. The combined dataset $\{D_1 \cup D_2\}$ contained the IP addresses of 5,266 unique Internet routers. The majority of the discovered routers belonged to the ISPs' networks (51%) and UUNET (45%). Moreover, our traces recorded approximately 200 routers that belonged to five additional Autonomous Systems (AS). The average end-to-end hop count was 11.3 in D_1 (6 minimum and 17 maximum) and 11.9 in D_2 (6 minimum and 22 maximum). Finally, the majority of paths (75% in D_1 and 65% in D_2) contained between 10 and 13 hops.

[5] Typical reasons for failing a session were PPP-layer connection problems, inability to reach the server (i.e., failed traceroute), high bit-error rates, low (14.4-19.2 kb/s) connection rates, and insufficient bandwidth.