# Measurement Study of Low-bitrate Internet Video Streaming

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

Hayder Radha
Dept. of Electrical & Computer Engineering
Michigan State University
East Lansing, MI 48824
radha@egr.msu.edu

***Abstract* – In this paper, we analyze the results of a seven-month real-time streaming experiment, which was conducted between a number of unicast dialup clients, connecting to the Internet through access points in more than 600 major U.S. cities, and a backbone video server. During the experiment, the clients streamed low-bitrate MPEG-4 video sequences from the server over paths with more than 5,000 distinct Internet routers. We describe the methodology of the experiment, the architecture of our NACK-based streaming application, study end-to-end dynamics of 16 thousand ten-minute sessions (85 million packets), and analyze the behavior of the following network parameters: packet loss, round-trip delay, one-way delay jitter, packet reordering, and path asymmetry. We also study the impact of these parameters on the quality of real-time streaming.**

## I. INTRODUCTION

The Internet has become a complex interconnection of a large number of computer networks. The behavior of the Internet has been the target of numerous studies, but nevertheless, the performance of the Internet from the perspective of an average home user still remains relatively undocumented. In addition, we believe that since end users are responsible for a large fraction of Internet traffic, the study of network conditions experienced by these users is an important research topic. These are the two main reasons that compelled us to conduct a fundamentally different performance study that looks at Internet dynamics from the angle of an average Internet user.

Even though the Internet has been extensively analyzed in the past, an overwhelming majority of previous studies were based on TCP traffic and involved high-speed bottleneck links. On the other hand, real-time streaming protocols have not received as much attention in these studies. Note that no previous work attempted to characterize the performance of real-time streaming in a large-scale Internet experiment of this sort (i.e., involving dialup access). The Internet has been studied from the perspective of TCP connections by Paxson [17], Bolliger *et al.* [4], Caceres *et al.* [8], Mogul [15], and several others (e.g., [3]). Paxson's study included 35 geographically distributed sites in 9 countries; Bolliger *et al.* employed 11 sites in 7 countries and compared the throughput performance of various implementations of TCP during a six-month experiment; whereas the majority of other researchers monitored transit TCP traffic at a single backbone router [3], [15] or inside several campus networks [8] for the duration ranging from several hours to several days.

The methodology used in both large-scale TCP experiments [4], [17] 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]

We believe that in order to study the current dynamics of real-time streaming in the Internet, we must take the same steps to connect to the Internet as an average end-user (i.e., through dialup ISPs). For example, ISPs often experience congestion in their own backbones, and during busy hours, V.90 (i.e., 56 kb/s) 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.

In addition to choosing a different topological setup for the experiment, our work is different from the previous studies in several other aspects. First, the sending rate of a TCP connection is driven by its congestion control, which can often cause increased packet loss and higher end-to-end delays in the path along which it operates (e.g., during slow start and congestion avoidance while probing for new bandwidth). In our experiment, we measured the end-to-end path dynamics as perceived by a CBR (constant bitrate) stream sharing backbone links with elastic TCP traffic. Hence, even though no active measurement study can avoid the effects of the Eisenberg principle, the use of CBR traffic allowed us to sample the real Internet performance without the bias of UDP-based congestion control applied to slow modem links.[2]

Second, TCP uses a positive ACK retransmission scheme, whereas real-time applications usually employ NACK-based retransmission to reduce the amount of traffic from the users to the streaming server. As a consequence, end-to-end path dynamics perceived by a NACK-based protocol could differ

[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] Without a doubt, future real-time streaming protocols will include some form of scalable congestion control; however, at the time of the experiment, it was not even clear which methods represented such congestion control.

from those sampled by TCP along the same path: real-time applications acquire samples of the round-trip delay (RTT) at rare intervals, send significantly less data along the path from the receiver to the sender, and bypass certain aspects of TCP's retransmission scheme (such as exponential timer backoff). Previous work [13] suggests that NACK-based retransmission schemes may require a different retransmission timeout (RTO) estimator and leads us to believe that research in this area should be extended.

We should further mention that the Internet has been extensively studied by various researchers using ICMP `ping` and `traceroute` packets [1], [9], [16], [17], [20], UDP echo packets [6], [7], and multicast backbone (MBone) audio packets [24]. With the exception of the last one, similar observations apply to these studies – neither the setup, nor the type of probe traffic represented realistic real-time streaming scenarios. In addition, among the studies that specifically sent video traffic over the Internet [5], [10], [21], [22], the majority of experiments involved only a few Internet paths, lasted for a very 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.

In this paper, we present the methodology and analyze the results of a seven-month large-scale real-time streaming experiment, which involved three nation-wide dialup ISPs, each with several million active subscribers in the United States. The topology of the experiment consisted of a backbone video server streaming MPEG-4 video sequences to unicast home users located in more than 600 major U.S. cities. The streaming was performed in real-time (i.e., with a real-time decoder), utilized UDP for the transport of all messages, and relied on simple NACK-based retransmission to recover lost packets before their decoding deadlines.

Even though we consider it novel and unique in many aspects, there is an important limitation to our study. Our experiments document Internet path dynamics perceived by low-bitrate (i.e., modem-speed) streaming sessions. Recall that one of the goals of our work was to conduct a performance study of the Internet from the angle of a typical home Internet user, and to this extent, we consider our work to be both thorough and successful. In addition, by focusing on low-bitrate paths, our study shows the performance of real-time protocols under the most difficult network conditions (i.e., large end-to-end delays, relatively high bit-error rates, low available bandwidth, etc.) and provides a "lower bound" on the performance of future Internet streaming applications.

Despite this limitation, we believe that the results of our study conclusively establish the feasibility of video streaming in the currently best-effort Internet and provide a valuable insight into dynamics of real-time streaming from the perspective of an average Internet user.

The remainder of the paper is organized as follows. Section II describes the methodology of the experiment. Section III provides an overview of the experiment. Section IV studies packet loss, section V analyzes the frequency of underflow events, section VI looks at the behavior of the round-trip delay, section VII briefly discusses one-way delay jitter, section VIII examines the extent of packet reordering during the experiment, and section IX studies the phenomenon of asymmetric paths discovered during the experiment. Section X concludes the paper.

## II. METHODOLOGY

### A. Setup for the Experiment

We started our work by attaching a Unix video server to the UUNET backbone via a T1 link (Figure 1). To support the client's 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) dialup numbers in the U.S., and designed an experiment in which hypothetical Internet users dialed a local access number to reach the Internet (through one of our three ISPs) and streamed video sequences from the server. Although the clients were physically placed in our lab in the state of New York, 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 1813 different V.90 access points in 1188 major U.S. cities.

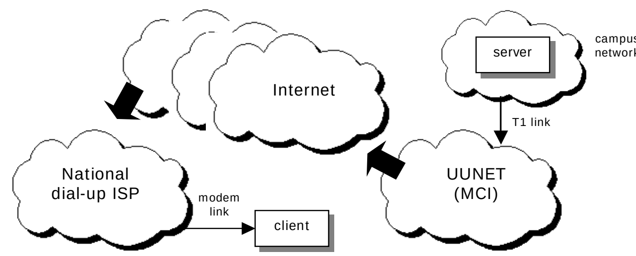


Figure 1. Topology of the experiment.

After the phone database was in place, we designed and implemented special software, which we call the *dialer*, that dialed phone numbers from the database, connected to the ISPs using the point-to-point protocol (PPP), issued a *parallel traceroute* to the server, and upon success, started the video client with the instructions to stream a ten-minute video sequence from the server. Our implementation of traceroute (built into the dialer) used ICMP probes, sent all probes in parallel instead of sequentially (hence the name "parallel"), and recorded the IP time-to-live (TTL)[3] field of each returned "TTL expired" message. The use of ICMP packets and parallel traceroute facilitated much quicker discovery of routers, and the analysis of the TTL field in the returned packets allowed the dialer to compute the number of hops in the reverse path from each intermediate router to the client machine (using a simple fact that each router reset the TTL field of each generated "TTL expired" packet to the value of the *initial*

[3] Recall that the TTL field is decremented by 1 every time a packet is forwarded by a level-3 (i.e., IP-level) device.

*TTL*[4]). Using the information about the number of *forward* and *reverse* hops for each router, the dialer was able to detect asymmetric end-to-end paths, which we study in section VIII.

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 (BER), and connections during which the modem could not sustain our streaming rates were all considered useless for our study and were excluded from the analysis in this paper.

In practice, to avoid studying connections with clearly insufficient end-to-end bandwidth and various modem-related problems, we utilized the following methodology. We defined a streaming attempt through a particular access number to be *successful*, if the ISP's access number was able to sustain the transmission of our video stream for its entire length at the stream's target IP bitrate $r$. To be specific, the video client terminated connections in which the *aggregate* (i.e., counting from the very beginning of a session) packet loss grew beyond a certain threshold $\beta_p$ or the *aggregate* incoming bitrate dropped below another threshold $\beta_r$. The experiments reported in this paper used $\beta_p$ equal to 15% and $\beta_r$ equal to $0.9r$, both of which were experimentally found to be necessary conditions for efficient filtering out of modem-related failures. The packet-loss threshold was activated after 1 minute of streaming and the bitrate threshold after 2 minutes to make sure that slight fluctuations in packet loss and incoming bitrate at the beginning of a session were not mistaken for poor connection quality. After a session was over, the success or failure of the session was communicated from the video client to the dialer, the latter of which kept track of the time of day and the phone number that either passed or failed the streaming test.

In order to make the experiment reasonably short, we considered all phone numbers from the same state to be equivalent, and consequently, we assumed that a successful streaming attempt through any phone number of a state indicated a *successful coverage* of the state regardless of which phone number was used. Furthermore, we divided each 7-day week into 56 three-hour timeslots (i.e., 8 timeslots per day) and designed the dialer to select phone numbers from the database in such order so that each state would be *successfully covered* within each of the 56 timeslots at least once. In other words, each ISP needed to sustain exactly $50\cdot 56 = 2{,}800$ successful sessions before the experiment was allowed to end.

### B. Real-time Streaming

For the purpose of the experiment, we used an MPEG-4 encoder to create two ten-minute QCIF (176x144) 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 experiment with stream $S_1$ lasted during November – December 1999 and the one with stream $S_2$ was an immediate follow-up during January – May 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 consequently, the last packet in each frame was allowed to be smaller than others (in fact, many P (prediction-coded) 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 special 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. Moreover, to simplify the analysis of the results, we implemented a *strict* real-time decoder model, in which the playback of the arriving frames continued at the encoder-specified deadlines regardless of the number of underflow events (i.e., the decoding deadlines were not adjusted based on network conditions).

In addition, many CBR (constant bitrate) video coding schemes include the notion of the *ideal startup delay* [18], [19] (the delay is called "ideal" because it assumes a network with no packet loss and a constant end-to-end delay). This ideal delay must always be applied to the decoder buffer before the decoding process may begin. The ideal startup delay is independent of the network conditions and solely depends on the decisions made by the encoder during the encoding process.[5] On top of this ideal startup delay, the client in a streaming session usually must apply an *additional* startup delay in order to compensate for delay jitter (i.e., variation in the one-way delay) and permit the recovery of lost packets via retransmission. This additional startup delay is called the *delay budget* ($D_{budget}$) and reflects the values of the *expected* (at the beginning of a session) delay jitter and round-trip delay during the length of the session. Note that in the context of Internet streaming, it is common to call $D_{budget}$ simply "startup delay" and to completely ignore the *ideal* startup delay (e.g., [10]). From this point on, we will use the same convention. In all our experiments, we used $D_{budget}$ equal to 2,700 ms, which was manually selected based on preliminary testing. Consequently, the total startup delay (observed by an

[4] The majority of routers used the initial TTL equal to 255, while some initialized the field to 30, 64, or 128. Subtracting the received TTL from the initial TTL produced the number of hops along the reverse path.

[5] We will not elaborate further on the ideal startup delay, except mention that it was approximately 1,300 ms for each stream.

end-user) at the beginning of each session was approximately 4 seconds.

### C. Client-Server Architecture

For the purpose of our experiment, we implemented a client-server architecture for MPEG-4 streaming over the Internet. The server was fully multithreaded to ensure that the transmission of packetized video was performed at the target IP bitrate of each streaming session and to provide quick response to clients' NACK requests. The streaming was implemented in bursts of packets (with the burst duration $D_b$ varying between 340 and 500 ms depending on the bitrate) for the purposes of making the server as low-overhead as possible (for example, RealAudio servers use $D_b$ = 1,800 ms [14]). Although we agree that in many cases the desired way of sending constant bitrate (CBR) traffic is to equally space packets during transmission, there are practical limitations (such as OS scheduling and inter-process switching delays) that often do not allow us to follow this model.

The second and the more involved part of our architecture, the client, was designed to recover lost packets through NACK-based retransmission and collect extensive statistics about each received packet and each decoded frame. Furthermore, as it is often done in NACK-based protocols, the client was in charge of collecting round-trip delay (RTT) samples (the resolution of timestamps was 100 microseconds). 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). In our experiment, in order to avoid the ambiguity of which retransmission of the same packet actually returned to the client, the header of each NACK request and each retransmitted packet contained an extra field specifying the retransmission number of the packet.

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 response to these simulated NACKs, the server included the usual overhead[6] of fetching the needed packets from the storage and sending them to the client. In our experiment, the client activated simulated NACKs, spaced 30 seconds apart, if packet loss was below 1%.

We tested the software 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. In addition to extensive testing of the prototype for nine months, we monitored various statistics reported by the clients in real-time (i.e., on the screen) during the experiments for sanity and consistency with previous tests.

Our traces consist of six datasets, each collected by a different machine. Throughout this paper, we will use notation $D_n^x$ to refer to the dataset collected by the client assigned to ISP$_x$ ($x$ = $a$, $b$, $c$) during the experiment with stream $S_n$ ($n$ = 1, 2). Furthermore, we will use notation $D_n$ to refer to the combined set $\{D_n^a \cup D_n^b \cup D_n^c\}$.

## III. 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.[7] 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. The success rate of streaming sessions during different times of the day is illustrated in Figure 2. Note the dip by a factor of two between the best and the worst times of the day.

Furthermore, in dataset $D_1$, the clients traced the arrival of 37.7 million packets, and in $D_2$, the arrival of additional 47.3 million (for a total of 85 million). In terms of bytes, the first experiment transported 9.4 GBytes of video data and the second one transported another 17.7 GBytes (for a total of 27.1 GBytes).

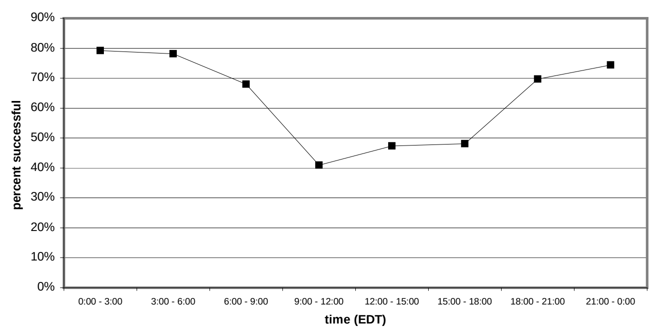


Figure 2. Success of streaming attempts during the day.

Recall that each experiment lasted as long as it was needed to cover the entire United States. Depending on the success rate within each state, the access points used in the experiment comprised a subset of our database. In $D_1$, the experiment covered 962 dialup points in 637 U.S. cities, and in $D_2$, it covered 880 dialup points in 575 U.S. cities. Figure 3 shows the combined (i.e., including both datasets $D_1$ and $D_2$) number of distinct cities in each state covered by our experiment (1003 access points in 653 cities).

[6] The server overhead was below 10 ms for all retransmitted packets and did not have a major impact on our characterization of the RTT process later in this paper.

[7] 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.

During the experiment, each session was preceded by a parallel traceroute, which recorded the IP addresses of all discovered routers (DNS and WHOIS[8] lookups were done off-line after the experiments were over). The average time needed to trace an end-to-end path was 1,731 ms, 90% of the paths were traced under 2.5 seconds, and 98% under 5 seconds. Dataset $D_1$ recorded 3,822 distinct Internet routers, $D_2$ recorded 4,449 distinct routers, and both experiments combined produced the IP addresses of 5,266 unique routers. The majority of the discovered routers belonged to the ISPs' networks (51%) and UUNET (45%), which confirmed our intuition before the experiment that all three ISPs had numerous (i.e., hundreds of) direct peering connections with UUNET (some of this information could also have been extracted from BGP tables). Moreover, our traces recorded approximately 200 routers that belonged to five additional Autonomous Systems (AS).

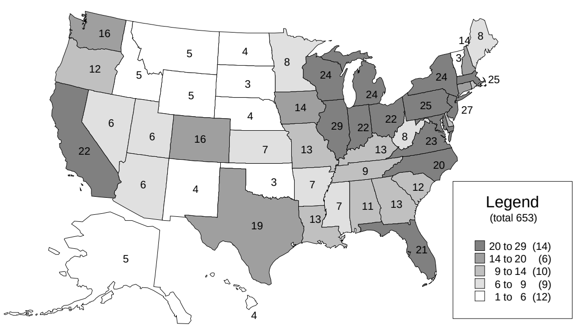


Figure 3. The number of cities per state that participated in either $D_1$ or $D_2$.

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). Figure 4 shows the distribution of the number of hops in the encountered end-to-end paths in each of $D_1$ and $D_2$. As the figure shows, the majority of paths (75% in $D_1$ and 65% in $D_2$) contained between 10 and 13 hops.

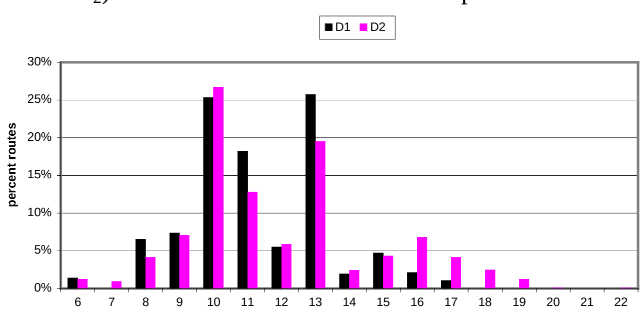


Figure 4. Distribution of the number of end-to-end hops.

Throughout the rest of the paper, we restrict ourselves to studying only *successful* (as defined in section II.A) sessions in both $D_1$ and $D_2$. We call these new purged datasets (with only successful sessions) $D_{1p}$ and $D_{2p}$, respectively (purged datasets $D_{np}^x$ are defined similarly for $n = 1, 2$ and $x = a, b, c$). Recall that $\{D_{1p} \cup D_{2p}\}$ contains 16,852 successful sessions, which are responsible for 90% of the bytes and packets, 73% of the routers, and 74% of the U.S. cities recorded in $\{D_1 \cup D_2\}$.

## IV. Packet Loss

### A. Overview

Numerous researchers have studied Internet packet loss, and due to the enormous diversity of the Internet, only few studies agree on the average packet loss rate and the average *loss burst length* (i.e., the number of packets lost in a row). Among numerous studies, the average Internet packet loss was reported to vary between 11% and 23% by Bolot [6] depending on the inter-transmission spacing between packets, between 0.36% and 3.54% by Borella *et al*. [7] depending on the studied path, between 1.38% and 11% by Yajnik *et al*. [24] depending on the location of the MBone receiver, and between 2.7% and 5.2% by Paxson [17] depending on the year of the experiment. In addition, 0.49% average packet loss rate was recently reported by Balakrishnan *et al*. [3], who analyzed the dynamics of a large number of TCP web sessions at a busy Internet server.

In dataset $D_{1p}$, the average recorded packet loss rate was 0.53% and in $D_{2p}$, it was 0.58%. Even though these rates are much lower[9] than those traditionally reported by Internet researchers during the last decade, they are still somewhat higher than those reported by backbone ISPs [23]. Furthermore, 38% of the sessions in $\{D_{1p} \cup D_{2p}\}$ did not experience any packet loss, 75% experienced loss rates below 0.3%, and 91% experienced loss rates below 2%. On the other hand, 2% of the sessions suffered packet loss rates 6% or higher.

In addition, as we expected, average packet loss rates exhibited a wide variation during the day. Figure 5 shows the evolution of loss rates as a function of the timeslot (i.e., the time of day), where each point represents the average of approximately 1,000 sessions. As the figure shows, the variation in loss rates between the best (3-6 am) and the worst (3-6 pm) times of the day was almost by a factor of four.

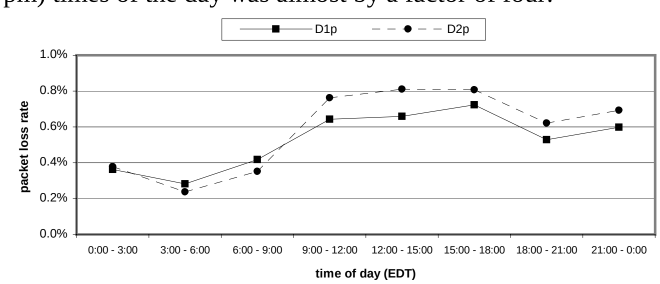


Figure 5. Average packet loss rates during the day.

[8] The WHOIS database was used to discover the Autonomous System (AS) of each router.

[9] Note that during the experiment, simply dialing a different access number in most cases fixed the problem of high packet loss. This fact shows that the majority of failed sessions documented pathologies created by the modem (or the access point) rather than the actual packet loss in the Internet. Since an end-user can always re-dial a bad connection searching for better network conditions, we believe that the bias created by removing failed sessions reflects the actions of a typical Internet user.

The apparent discontinuity in Figure 5 between timeslots 7 (21:00-0:00) and 0 (0:00-3:00) is due to the coarse timescale used in the figure. On finer timescales (e.g., minutes), loss rates converge to a common value near midnight. A similar discontinuity in packet loss rates was reported by Paxson [17] for North American sites, where packet loss during timeslot 7 was approximately twice as high as that during timeslot 0.

The variation in the average *per-state* packet loss (as shown in Figure 6) was quite substantial (from 0.2% in Idaho to 1.4% in Oklahoma), but virtually did not depend on the state's average number of end-to-end hops (correlation coefficient ρ was –0.04) or the state's average RTT (correlation –0.16). However, as we will see later, the average per-state RTT and the number of end-to-end hops were in fact positively correlated.

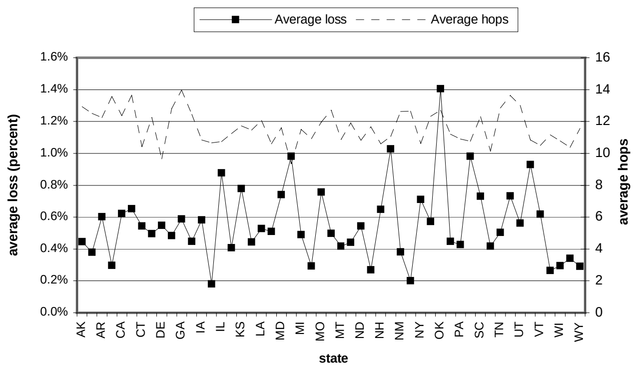


Figure 6. Average per state packet loss rates.

### B. *Loss Burst Lengths*

We next attempt to answer the question of how bursty Internet packet loss was during the experiment. Figure 7 shows the distribution (both the PDF and the CDF) of loss burst lengths in $\{D_{1p} \cup D_{2p}\}$ (without loss of generality, the figure stops at burst length 20, covering more than 99% of the bursts). Even though the upper tail of the distribution had very few samples, it was fairly long and reached burst lengths of over 100 packets.

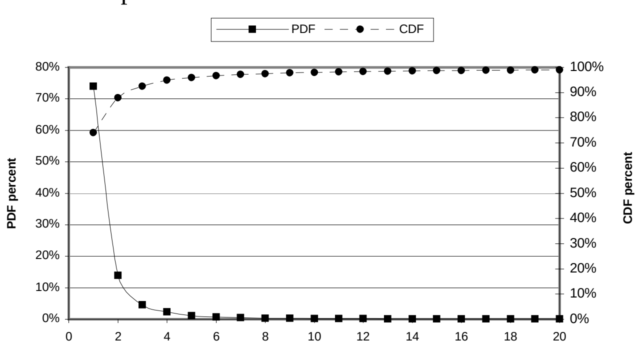


Figure 7. PDF and CDF functions of loss burst lengths in $\{D_{1p} \cup D_{2p}\}$.

Figure 7 is based on 207,384 loss bursts and 431,501 lost packets. The prevalence of single packet losses, given the fact that packets in our experiment were injected into the Internet in bursts at the T1 speed[10], leads us to speculate that either router queues sampled in our experiment overflowed on timescales smaller than the time needed to transmit a single IP packet over a T1 link (i.e., 3 ms for the largest packets and 1.3 ms for the average-size packets), or that backbone routers employed Random Early Detection (RED) for preventing congestion.

To investigate the presence of RED in the Internet, we contacted several backbone and dialup ISPs whose routers were recorded in our trace data and asked them to comment on the deployment of RED in their backbones. Among the ISPs that responded to our request, the majority had purposely disabled RED and the rest were running RED only for select customers at border routers, but not on the public backbone.

Ruling out RED, a second look at the situation reveals that server's possible interleaving of packets from different sessions could have expanded the inter-packet transmission distance of each flow by up to a factor of 3. Furthermore, before each lost packet reached the corresponding congested router, packets from other Internet flows could have queued immediately behind the lost packet, effectively expanding the inter-packet distance even further. Therefore, even though our data point toward transient buffer overflow events during the experiment (i.e., 1-3 ms), this conclusion may not necessarily hold in all cases.

Furthermore, as previously pointed out by many researchers (e.g., [24]), the upper tail of loss burst lengths usually contains a substantial percentage of all lost packets. In each of $D_{1p}$ and $D_{2p}$, single-packet bursts contained only 36% of all lost packets, bursts two packets or shorter contained 49%, bursts 10 packets or shorter contained 68%, and bursts 30 packets or shorter contained 82%. At the same time, 13% of all lost packets were dropped in bursts at least 50 packets long.

Traditionally, the burstiness of packet loss is measured by the average loss burst length. In the first dataset ($D_{1p}$), the average burst length was 2.04 packets. In the second dataset ($D_{2p}$), the average burst length was slightly higher (2.10), but not high enough to conclude that the higher bitrate of stream $S_2$ was clearly responsible for burstier packet loss. Furthermore, the *conditional probability* of packet loss, given that the previous packet was also lost, was 51% in $D_{1p}$ and 53% in $D_{2p}$. These numbers are consistent with those previously reported in the literature. Bolot [6] observed the conditional probability of packet loss to range from 18% to 60% depending on inter-packet spacing during transmission, Borella *et al.* [7] from 10% to 35% depending on the time of day, and Paxson [17] reported 50% conditional probability for *loaded* (i.e., queued behind the previous) TCP packets and 25% for *unloaded* packets. Using Paxson's terminology, the majority

[10] The server was only involved in low-bitrate streaming for our clients and did not have a problem blasting bursts of packets at the full speed of the adjacent link (i.e., 10 mb/s). The spacing between packets was further expanded by the T1 link to UUNET.

of our packets were *loaded* since the server sent packets in bursts at a rate higher than the bottleneck link's capacity.

### C. Loss Burst Durations

To a large degree, the average loss burst length depends on how closely the packets are spaced during transmission. Assuming that bursty packet loss comes from buffer overflow events in drop-tail queues rather than from consecutive hits by RED or from bit-level corruption, it is clear that all packets of a flow passing through an overflown router queue will be dropped for the duration of the instantaneous congestion. Hence, the closer together the flow's packets arrive to the router, the more packets will be dropped during each queue overflow. This fact was clearly demonstrated in Bolot's experiments [6], where UDP packets spaced 8 ms apart suffered larger loss burst lengths (mean 2.5 packets) than packets spaced 500 ms apart (mean 1.1 packets). Yajnik *et al.* [24] reported a similar correlation between loss burst lengths and the distance between packets. Consequently, instead of analyzing burst lengths, one might consider analyzing burst durations since the latter does not depend on inter-packet spacing during transmission.

Using our traces, we can only infer an approximate duration of each loss burst, because we do not know the *exact* time when the lost packets were supposed to arrive to the client (i.e., knowing the *exact* arrival time would require the knowledge of delay jitter experienced by the lost packet; however, since they were lost, this information is unavailable). Hence, for each loss event, we define the *loss burst duration* as the time elapsed between the receipt of the packet immediately preceding the loss burst and the packet immediately following the loss burst. Figure 8 shows the distribution (CDF) of loss burst durations in seconds. Although the distribution tail is quite long (up to 36 seconds), the majority (more than 98%) of loss burst durations in both datasets $D_{1p}$ and $D_{2p}$ fall under 1 second. Paxson's study [17] also observed large loss burst durations (up to 50 seconds), however, only 60% of the loss bursts studied by Paxson were contained below 1 second. In addition, our traces showed that the average distance between lost packets in the experiment was 172-188 good packets, or 21-27 seconds, depending on the streaming rate.

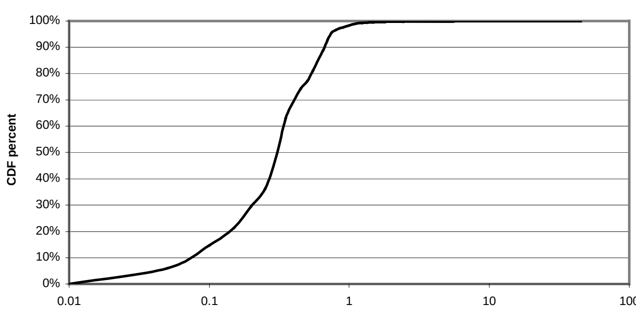


Figure 8. The CDF function of loss burst durations in $\{D_{1p} \cup D_{2p}\}$.

### D. Heavy Tails

In conclusion of this section, it is important to note that packet losses sometimes cannot be modeled as independent events due to buffer overflows that last long enough to affect multiple adjacent packets. Consequently, future real-time protocols should expect to deal with bursty packet losses (Figure 7) and possibly heavy-tailed distributions of loss burst lengths (see below).

Several researchers reported a heavy-tailed nature of loss burst lengths, and the shape parameter $\alpha$ of the Pareto distribution fitted to the length (or duration) of loss bursts was recorded to range from 1.06 (Paxson [17]) to 2.75 (Borella *et al.* [7]). On the other hand, Yajnik *et al.* [24] partitioned the collected data into stationary segments and reported that loss burst lengths could be modeled as exponential (i.e., not heavy-tailed) within each stationary segment. In addition, Zhang *et al.* [25] reported that packet loss along some Internet paths was stationary and could be modeled as exponential, whereas other paths were found to be non-stationary and not easy to model.

Using intuition, it is clear that packet loss and RTT random processes in both $D_{1p}$ and $D_{2p}$ are expected to be non-stationary. For example, the non-stationarity can be attributed to the time of day or the location of the client. In either case, we see three approaches to modeling such non-stationary data. In the first approach, we would have to analyze 16,852 PDF functions (one for each session) for stationarity and heavy tails. Unfortunately, an average session contained only 24 loss bursts, which is insufficient to build a good distribution function for a statistical analysis.

The second approach would be to combine all sessions into groups, which are intuitively perceived to be stationary (e.g., according to the access point or the timeslot), and then perform similar tests for stationarity and heavy tails within each group. We might consider this direction for future work as it appears to be the right way of modeling random variables sampled over diverse Internet paths and during different times of the day.

The third approach is to do what the majority has done in the past – assume that all data samples belong to a stationary process and are drawn from a single distribution. Using this last approach, Figure 9 shows a log-log plot of the complementary CDF function from Figure 7 with a least-squares fit of a straight line representing a hyperbolic (i.e., heavy-tailed) distribution (the dotted curve is the exponential distribution fitted to the data). The fit of a straight line is quite good (with correlation $\rho = 0.99$) and provides a strong indication that the distribution of loss burst lengths in the combined dataset $\{D_{1p} \cup D_{2p}\}$ is heavy-tailed. Furthermore, as expected, we notice that the exponential distribution in Figure 9 decays too quickly to even remotely fit the data.

Finally, consider a Pareto distribution with a CDF $F(x) = 1-(\beta/x)^{\alpha}$ and PDF $f(x) = \alpha\beta^{\alpha}x^{-\alpha-1}$, where $\alpha$ is the shape parameter and $\beta$ is the location parameter. Using Figure 9, we establish that a Pareto distribution with $\alpha = 1.34$ (finite mean, but infinite variance) and $\beta = 0.65$ fits our data very well.

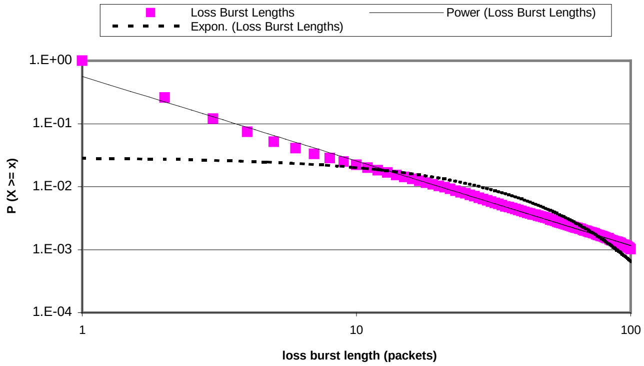


Figure 9. The complimentary CDF of loss burst lengths in $\{D_{1p} \cup D_{2p}\}$ on a log-log scale fitted with hyperbolic (straight line) and exponential (dotted curve) distributions.

## V. UNDERFLOW EVENTS

The impact of packet losses on real-time applications is understood fairly well. Note that in this paper, we use a fairly simplified (but nevertheless very realistic) assumption that each video frame with a least one lost packet causes an *underflow event*. There are several reasons for doing so. First, most compression protocols (including MPEG-4) do not standardize error concealment techniques that are aimed at concealing the loss of an entire packet (i.e., at most, they recover from bit errors). Packet loss concealment techniques vary from obvious to extremely complex, and we tried to avoid turning this paper into an evaluation study of any particular concealment technique. Second, the performance of error concealment varies greatly depending on the amount of motion in the video, the location of the lost packet (i.e., I, P, or B frame), and the compression method used. Furthermore, in many cases, concealing a loss of an entire packet is simply impossible (i.e., when the bitrate is low, and especially cases when the entire frame fits into a single packet). Consequently, this paper does not consider error concealment in conjunction with packet loss, and each video frame with missing packets at the time of decoding results in an underflow event.

In addition, this paper does not discuss any video quality related metrics (e.g., peak signal to noise ratio (PSNR)) of the received stream. Even though the user is ultimately interested in the video quality of the stream, the video performance depends on many different parameters (i.e., encoder options, GOP structure, the amount of motion in the video, etc.) and provides little information about the actual network conditions in the Internet.

We should further mention that all groups of pictures (GOPs) in both video streams were composed of I and P frames, which means that a loss of a single frame propagated until the end of the corresponding GOP (i.e., a single underflow event typically caused several other underflow evens in the same GOP). Note that somewhat better performance can be achieved if the video stream is coded with B frames; however, the analysis of such streams is impossible without conducting a brand new experiment (i.e., it is beyond the scope of this paper).

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.

Consider the following. In $\{D_{1p} \cup D_{2p}\}$, 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 a very efficient method of overcoming packet loss in real-time applications. Clearly, the success rate will be even higher in networks with smaller end-to-end delays.

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 (i.e., partially late packets were used to "rescue" the remaining frames from the same GOP). Out of 9,013 late retransmissions, 4042 (49%) were partially late. Using each partially late packet, the client was able to "rescue" on average 4.98 frames from the same GOP[11] in $D_{1p}$ and 4.89 frames in $D_{2p}$ by employing the above-described *catch-up* decoding technique (for more discussion, see [20]).

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 +

[11] We used 10-frame GOPs in both sequences.

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 $D_{1p}$ and 8.6 seconds in $D_{2p}$, which can be considered excellent given the small amount of delay budget (i.e., startup delay) used in the experiments.

To further understand the phenomenon of late packets, we plotted in Figure 10 the CDFs of the amount of time by which late packets missed their deadlines (i.e., the amount of time that we need to add to delay budget $D_{budget}$ = 2,700 ms 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 $D_{budget}$ 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). Hence, for example, Internet applications utilizing a 13-second delay budget (which corresponds to 10.3 seconds of *additional* delay in Figure 10) would be able to recover 99% of retransmissions and 84% of data packets that were late during the experiment, given similar streaming conditions.

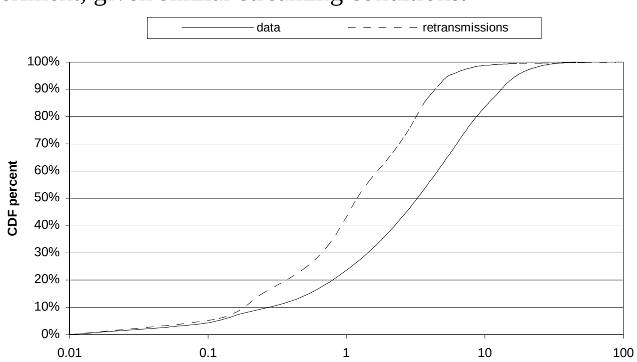


Figure 10. CDF functions of the amount of time by which retransmitted and data packets were late for decoding.

## VI. ROUND-TRIP DELAY

### A. Overview

Figure 11 shows the PDF functions of the round-trip delay in each of $D_{1p}$ and $D_{2p}$ (660,439 RTT samples in both datasets). Although the tail of the combined distribution reached the enormous values of 126 seconds for *simulated* and 102 seconds for *real* retransmissions, the majority (75%) of the samples were below 600 ms, 90% below 1 second, and 99.5% below 10 seconds. The average RTT was 698 ms in $D_{1p}$ and 839 ms in $D_{2p}$. The minimum RTT was 119 and 172 ms, respectively. Although very rare, extremely high RTTs were found in all six datasets $D_{1p}^{a} – D_{2p}^{c}$ (as a reminder, dataset $D_{np}^{x}$ consists of successful sessions involving stream $S_n$ and recorded through $ISP_x$). Furthermore, out of more than 660 thousand RTT samples in $\{D_{1p} \cup D_{2p}\}$, 437 were at least 30 seconds, 32 at least 50 seconds, and 20 at least 75 seconds.

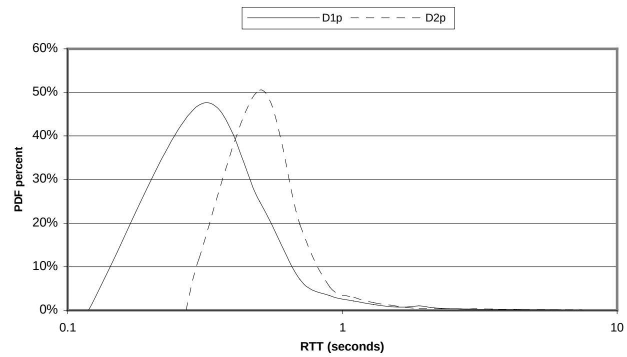


Figure 11. PDF functions of the RTT samples in each of $D_{1p}$ and $D_{2p}$.

Although pathologically high RTTs may seem puzzling at first, there is a simple explanation. Modem error correction protocols (i.e., the commonly used V.42) implement retransmission for corrupted blocks of data on the physical layer.[12] Error correction is often necessary if modems negotiated data compression (i.e., V.42bis) over the link and is desirable if PPP Compression Control Protocol (CCP) is enabled on the data-link layer. In all our experiments, both types of compression were enabled, imitating the typical setup of a home user. Therefore, if a client established a connection to a remote modem at a low bitrate (which was sometimes accompanied by a significant amount of noise in the phone line), each retransmission on the physical layer took a large time to complete before the data was delivered to the upper layers. In addition, large IP-level buffers on either side of the modem link further delayed packets arriving to or originating from the client host.

Note that the purpose of classifying sessions into failed and successful in section II.A was to avoid reporting pathological conditions caused by the modem links. Since only a handful (less than 0.5%) of RTTs in $\{D_{1p} \cup D_{2p}\}$ were seriously effected by modem-level retransmission and bit errors (we consider sessions with RTTs higher than 10 seconds to be caused by modem-related problems[13]), we conclude that our heuristic

[12] 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.

[13] For example, one of the authors uses a modem access point at home with IP-level buffering on the ISP side equivalent to 6.7 seconds. Consequently, delays as high as 5-10 seconds may often be caused by non-pathological conditions.

was successful in filtering out the majority of pathological connections and that future application-layer protocols, running over a modem link, should be prepared to experience RTTs in the order of several seconds.

Furthermore, the removal of sessions with RTTs higher than 10 seconds does not change any of the results below and, at the same time, prohibits us from showing the extent of variation in the network parameters experienced by a home Internet user. Therefore, since all sessions in $\{D_{1p} \cup D_{2p}\}$ were able to successfully complete, we consider the removal of sessions based on their large RTT to be unwarranted.

### B. Heavy Tails

Mukherjee [16] 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 in Figure 11 resembles that of a gamma function, the distribution tails in the figure decay much slower than those of an exponential distribution (see below).

Using our approach from section IV.D (i.e., assuming that each studied Internet random process is stationary), we extracted the upper tails of the PDF functions in Figure 11 and plotted the results on a log-log scale in Figure 12. The figure shows that a straight line (without loss of generality fitted to the PDF of $D_{2p}$ in the figure) provides a good fit to the data (correlation 0.96) and allows us to model the upper tails of the PDF functions as a Pareto distribution with PDF $f(x) = \alpha\beta^{\alpha}x^{-\alpha-1}$, where shape parameter $\alpha$ equals 1.16 in dataset $D_{1p}$ and 1.58 in $D_{2p}$ (as before, the distribution has a finite mean, but an infinite variance).

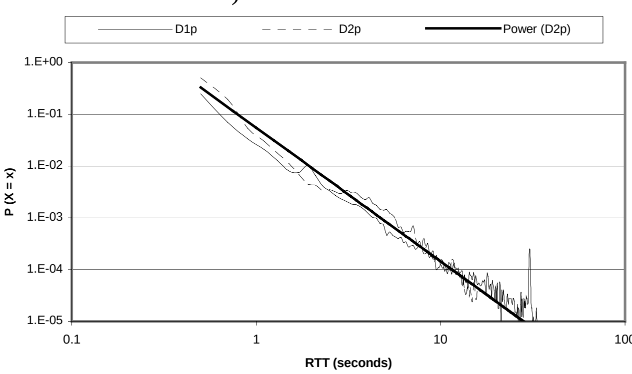


Figure 12. Log-log plot of the upper tails of the distribution of the RTT (PDF). The straight line is fitted to the PDF from $D_{2p}$.

### C. Variation of the RTT

We conclude the discussion of the RTT by showing that the round-trip delay exhibited a variation during the day similar to that of packet loss (previously shown in Figure 5) and that the average RTT was positively correlated with the length of the end-to-end path. Figure 13 shows the average round-trip delay during each of eight timeslots of the day (as before, each point in the figure represents the average of approximately 1,000 sessions). The figure confirms that the worst time for sending traffic over the Internet is between 9 am and 6 pm EDT and shows that the increase in the delay during the peak hours is relatively small (i.e., by 30-40%).

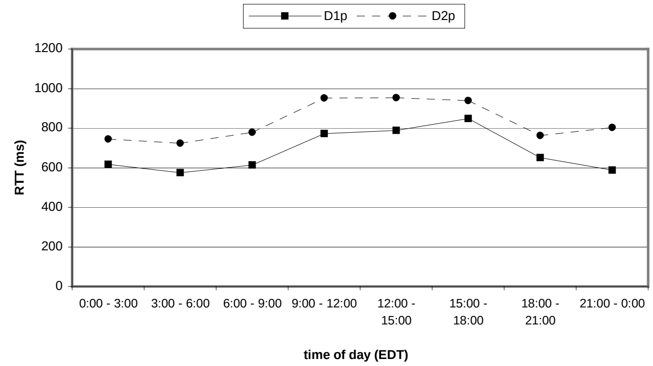


Figure 13. Average RTT as a function of the time of day.

Figure 14 shows the average RTT sampled by the clients in each of the 50 U.S. states. The average round-trip delay was consistently high (above 1 second) for three states – Alaska, New Mexico, and Hawaii. On the other hand, the RTT was consistently low (below 600 ms) also for three states – Maine, New Hampshire, and Minnesota. These results (except Minnesota) can be directly correlated with the distance from New York; however, in general, we find that the geographical distance of the access point from the East Coast had little correlation with the average RTT. Thus, for example, some states in the Midwest had small (600-800 ms) average round-trip delays and some states on the East Coast had large (800-1000 ms) average RTTs.

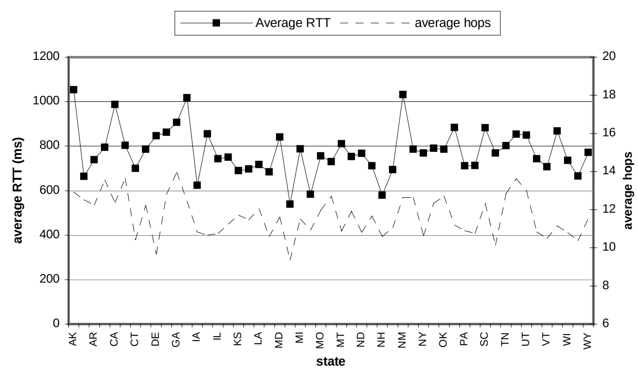


Figure 14. Average RTT and average hop count in each of the states in $\{D_{1p} \cup D_{2p}\}$.

A more substantial link can be established between the number of end-to-end hops and the average RTT as shown in Figure 14. Even though the average RTT of many states did not exhibit a clear dependency on the average length of the path, the correlation between the RTT and the number of hops in Figure 14 was reasonably high with $\rho = 0.52$. This result was intuitively expected since the RTT is essentially the sum of queuing and transmission delays at intermediate routers.

## VII. Delay Jitter

As we discussed above, in certain streaming situations, round-trip delays are much less important to real-time applications than one-way delay jitter, because the latter can potentially cause significantly more underflow events. In addition, due to asymmetric path conditions (i.e., uneven congestion in the upstream and downstream directions), large RTTs are not necessarily an indication of bad network conditions for a NACK-based application. For example, in many sessions with high RTTs during the experiment, the outage was caused by the upstream path, while the downstream path did not suffer from extreme one-way delay variation, and data (i.e., non-retransmitted) packets were arriving to the client throughout the entire duration of the outage. Hence, we conclude that the value of the RTT was not necessarily a good indicator of a session's quality during the experiment and that one-way delay jitter should be used instead.

Assuming that delay jitter is defined as the difference between one-way delays of each two consecutively sent packets, an application can sample both positive and negative values of delay jitter. Negative values are produced by two types of packets – those that suffered a *packet compression event* (i.e., the packets' arrival spacing was smaller than their transmission spacing) and those that became reordered. The former case is of great interest in packet-pair bandwidth estimation studies and otherwise remains relatively unimportant. The latter case will be studied in section VIII under packet reordering. On the other hand, positive values of delay jitter represent *packet expansion events*, which are responsible for late packets. Consequently, we analyzed the distribution of only *positive* delay jitter samples and found that although the highest sample was 45 seconds, 97.5% of the samples were under 140 ms and 99.9% under 1 second. As the above results show, 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 in previous sections, even though the overall delay jitter was relatively low.

## VIII. Packet Reordering

### A. Overview

Real-time protocols often rely on the assumption that packet reordering in the Internet is a rare and an insignificant event for all practical purposes (e.g., [10]). Although this assumptions simplifies the design of a protocol, it also make the protocol poorly suited for the use over the Internet. Certainly, there are Internet paths along which reordering is either non-existent or extremely low. At the same time, there are paths that are dominated by multipath routing effects and often experience reordering (e.g., Paxson [17] reported a session with 36% of packets arriving out of order).

Unfortunately, there is not much data documenting reordering rates experienced by IP traffic over modem links. Using intuition, we expected reordering in our experiments to be extremely rare given the low bitrates of streams $S_1$ and $S_2$. However, we were surprised to find out that certain paths experienced consistent reordering with a relatively large number of packets arriving our of order, although the average reordering rates in our experiments were substantially lower than those reported by Paxson [17].

For example, in dataset $D_{1p}{}^a$, we observed that approximately out of every three missing[14] packets one was reordered. Hence, if users of ISP$_a$ employed a streaming protocol, which used a gap-based detection of lost packets [10] (i.e., the first out-of-order packet triggers a NACK), one third of NACKs would be flat-out redundant and a large number of retransmissions would be unnecessary, causing a noticeable fraction of ISP's bandwidth to be wasted.

Since each missing packet is potentially reordered, the true frequency of reordering can be captured by computing the percentage of reordered packets relative to the total number of *missing* packets. The average reordering rate in our experiment was 6.5% of the number of *missing* packets, or 0.04% of the number of *sent* packets. These numbers show that our reordering rates were at least by a factor of 10 lower than those reported by Paxson [17], whose average reordering rates varied between 0.3% and 2% of number of sent packets depending on the dataset. This difference can be explained by the fact that our experiment was conducted at substantially lower end-to-end bitrates, as well as by the fact that Paxson's experiment involved several paths with extremely high reordering rates.

Out of 16,852 sessions in $\{D_{1p} \cup D_{2p}\}$, 1,599 (9.5%) experienced at least one reordering. Interestingly, the average session reordering rates in our datasets were very close to those in Paxson's 1995 data [17] (12% sessions with at least one reordering), despite the fundamental differences in sending rates. The highest reordering rate *per ISP* in our experiment occurred in $D_{1p}{}^a$, where 35% of the number of missing packets (0.2% of the number of sent packets) turned out to be reordered. In the same $D_{1p}{}^a$, almost half of the sessions (47%) experienced at least one reordering event. Furthermore, the maximum number of reordered packets in a single session occurred in $D_{1p}{}^b$ and was 315 packets (7.5% of the number of sent packets).

Interestingly, the reordering probability did not show any dependence on the time of day (i.e., the timeslot), and was virtually the same for all states.

### B. Reordering Delay

To further study packet reordering, we define two metrics that allow us to measure the extent of packet reordering. First, let *packet reordering delay* $D_r$ be the delay from the time when a reordered packet was declared as *missing* to the time when the reordered packet arrived to the client. Second, let

[14] Missing packets are defined as gaps in sequence numbers.

*packet reordering distance* $d_r$ be the number of packets (including the very first out-of-sequence packet, but not the reordered packet itself) received by the client during reordering delay $D_r$. These definitions are illustrated in Figure 15, where reordering distance $d_r$ is 2 packets and reordering delay $D_r$ is the delay between receiving packets 3 and 2.

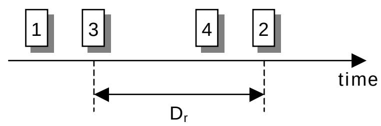


Figure 15. The meaning of reordering delay $D_r$.

Figure 16 shows the PDF of the reordering delay $D_r$ in $\{D_{1p} \cup D_{2p}\}$. The largest reordering distance $d_r$ in the combined dataset was 10 packets, and the largest reordering delay $D_r$ was 20 seconds (however, in the latter case, $d_r$ was only 1 packet). Although quite large, the maximum value of $D_r$ is consistent with previously reported numbers (e.g., 12 seconds in Paxson's data [17]). The majority (90%) of samples in Figure 16 are below 150 ms, 97% below 300 ms, and 99% below 500 ms.

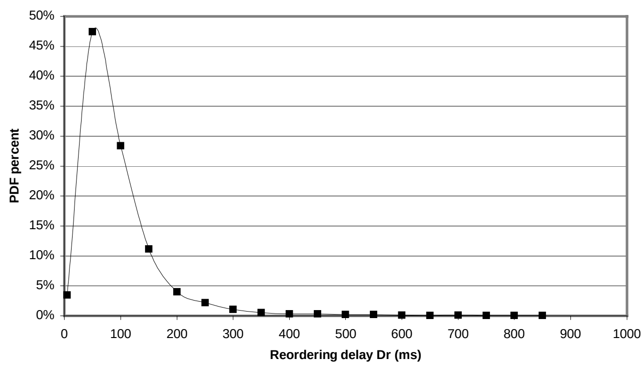


Figure 16. The PDF of reordering delay $D_r$ in $\{D_{1p} \cup D_{2p}\}$.

### *C. Reordering Distance*

We next analyze the suitability of TCP's triple-ACK scheme in helping NACK-based protocols detect reordering. TCP's *fast retransmit* relies on *three* consecutive duplicate ACKs (hence the name "triple-ACK") from the receiver to detect packet loss and avoid unnecessary retransmissions. Therefore, if reordering distance $d_r$ is either 1 or 2, the triple-ACK scheme successfully avoids duplicate packets, and if $d_r$ is greater than or equal to 3, it generates a duplicate packet. Figure 17 shows the PDF of reordering distance $d_r$ in both datasets. Using the figure, we can infer that TCP's triple-ACK would be successful for 91.1% of the reordering events in our experiment, double-ACK for 84.6%, and quadruple-ACK for 95.7%. Note that Paxson's TCP-based data [17] show similar, but slightly better detection rates, specifically 95.5% for triple-ACK, 86.5% for double-ACK, and 98.2% for quadruple-ACK.

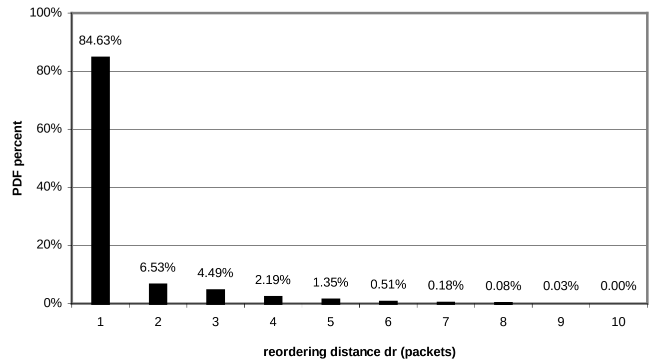


Figure 17. The PDF of reordering distance $d_r$ in $\{D_{1p} \cup D_{2p}\}$.

## IX. ASYMMETRIC PATHS

Recall that during the initial executions of traceroute, our dialer recorded the TTL fields of each received "TTL expired" packet. The TTL fields of these packets allowed the dialer to compute the number of hops between the router that generated a particular "TTL expired" message and the client. Suppose some router $i$ was found to be located at hop $f_i$ in the *upstream* (i.e., forward) direction and at hop $r_i$ in the *downstream* (i.e., reverse) direction. Hence, we can conclusively establish that an $n$-hop end-to-end path is *asymmetric*, if there exists a router for which the number of downstream hops is different from the number of upstream hops (i.e., $\exists i, 1 \le i \le n: f_i \ne r_i$). However, the opposite is not always true – if each router has the same number of downstream and upstream hops, we cannot conclude that the path is symmetric (i.e., it could be asymmetric as well). Hence, we call such paths *probably-symmetric*.[15]

In $\{D_{1p} \cup D_{2p}\}$, 72% of the sessions sent their packets over *definitely* asymmetric paths. To further understand how path asymmetry depends on the number of end-to-end hops, we extracted path information from $\{D_{1p} \cup D_{2p}\}$ and counted each end-to-end path through a particular access point exactly once. Figure 18 shows the percentage of asymmetric paths as a function of the number of end-to-end hops in the path. As the figure shows, almost all paths with 14 hops or more were asymmetric, as well as that even the shortest paths (with only 6 hops) were prone to asymmetry. This result can be explained by the fact that longer paths are more likely to cross over AS boundaries or intra-AS administrative domains. In both cases, "hot-potato" routing policies can cause path asymmetry.

Clearly, this result depend on the particular ISP employed by the end-user and the autonomous systems that user traffic traverses. Large ISPs (such as the ones studied in this work) often employ numerous peering points (hundreds in our case), and the results shown in Figure 18 may not hold for smaller ISPs. Nevertheless, our data show that dialup home

[15] In the most general case, even performing a reverse traceroute from the server to the client could not have conclusively established each path's symmetry (because the "TTL expired" messages identify router *interfaces* rather than individual *routers*). See [17] for more discussion.

users in general are more likely to experience an asymmetric end-to-end path than a symmetric one.

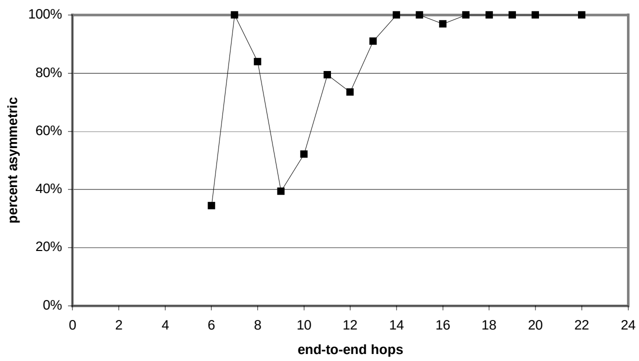


Figure 18. Percentage of asymmetric routes in $\{D_{1p} \cup D_{2p}\}$ as a function of the number of end-to-end hops.

## X. Conclusion

Our experiments with real-time streaming in the Internet indicate the following:

- Internet packet loss is bursty, and the distribution of loss burst lengths and the RTT appear to be heavy-tailed;
- one-way delay jitter appears to be much more harmful to low-bitrate real-time applications than either packet loss or large RTTs;
- in the current Internet, RTTs in the order of several seconds are possible along paths with a slow modem link;
- the average RTT appears to be positively correlated with the number of end-to-end hops, whereas packet loss does not seem to depend on the number of hops or the average RTT along the same path;
- in the current Internet, packet reordering can be experienced even by the paths with very slow bottleneck links, however, both the reordering delay and the reordering distance are relatively small;
- the majority of Internet paths sampled by our experiment were asymmetric, and a path's asymmetry could be linked to the path's length.

## References

[1] A. Acharya, and J. Saltz, "A Study of Internet Round-trip Delay," *Technical Report CS-TR-3736*, University of Maryland, December 1996.

[2] M. Allman, and V. Paxson, "On Estimating End-to-End Network Parameters," *ACM SIGCOMM*, September 1999.

[3] H. Balakrishnan, V.N. Padmanablah, S. Seshan, M. Stemm, and R.H. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements," *IEEE INFOCOM*, March 1998.

[4] J. Bolliger, T. Gross, and U. Hengartner, "Bandwidth Modelling for Network-Aware Applications," *IEEE INFOCOM*, March 1999.

[5] J-C. Bolot, and T. Turletti, "Experience with Rate Control Mechanisms for Packet Video in the Internet," *ACM Computer Communication Review*, January 1998 (*earlier paper in IEEE INFOCOM, 1994*).

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

[7] M.S. Borella, D. Swider, S. Uludag, and G.B. Brewster, "Internet Packet Loss: Measurement and Implications for End-to-End QoS," *International Conference on Parallel Processing*, August 1998.

[8] R. Caceres, P.B. Danzig, S. Jamin, D. Mitzel, "Characteristics of Wide-Area TCP/IP Conversations," *ACM SIGCOMM*, 1991.

[9] K. Claffy, G.C. Polyzos, and H-W. Braun, "Traffic Characteristics of the T1 NSFNET Backbone," *IEEE INFOCOM*, 1993.

[10] B. Dempsey, J. Liebeherr, and A. Weaver, "On retransmission-based error control for continuous media traffic in packet-switching networks," *Computer Networks and ISDN Systems*, vol. 28, no. 5, March 1996.

[11] S. Floyd, M. Handley, and J. Padhye, "Equation-Based Congestion Control for Unicast Applications," *ACM SIGCOMM*, September 2000.

[12] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM*, 1988.

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

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

[15] J.C. Mogul, "Observing TCP Dynamics in Real Networks," *ACM SIGCOMM*, 1992.

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

[17] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," *Ph.D. dissertation*, Computer Science Department, University of California at Berkeley, 1997.

[18] H. Radha, Y. Chen, K. Parthasarathy, and R. Cohen, "Scalable internet video using MPEG-4," *Signal Processing: Image Communication*, 1999.

[19] A.R. Reibman, and B.G. Haskell, "Constrains on Variable Bitrate Video for ATM Networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 4, December 1992.

[20] I. Rhee, "Error Control Techniques for Interactive Low Bitrate Video Transmission over the Internet," *ACM SIGCOMM*, September 1998.

[21] 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.

[22] T. Turletti, and C. Huitema, "Videoconferencing Over the Internet," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, June 1996.

[23] UUnet Latency Statistics. *http://uunet.com/network/latency*.

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

[25] Y. Zhang, V. Paxson, and S. Shenker, "The Stationarity of Internet Path Properties: Routing, Loss, and Throughput," *ACIRI Technical Report*, May 2000, http://www.aciri.org/vern/papers.html.