Abstract

We take the position that it is time to revisit and challenge the dogma that TCP is undesirable for audio and video streaming. We contend that new compression practices and reduced storage costs make TCP a viable and attractive basis for streaming stored content. Our approach has much in common with the recent proliferation of work on TCP-friendly streaming, but using TCP itself poses distinct challenges and provides several advantages. To support our position, this paper describes an architecture for content preparation and delivery intended to demonstrate effective streaming of stored content over TCP. We present preliminary results from implementing a QoS-adaptive video system based on the proposed architecture, and describe our ongoing work on streaming.

1 Introduction

Lossy compression is an old idea for digital video. The basic principal is to allow dropping of data in exchange for higher compression ratios. Lossy compression has been deeply studied, and many heuristics for how to best drop data are now well known. These heuristics often work by taking advantage of characteristics of human perceptual systems. Lossy compression techniques are usually judged through rate-distortion metrics. Rate-distortion metrics are measures which relate the compression ratio to some objective measure of quality loss. Much of the progress in the field has concerned improving rate-distortion performance. While the rate-distortion trade-off has always been fundamental to lossy video compression work, it is only recently that common video compression standards have added support for adjustment of the trade-off during the delivery step rather than just during encoding step[8, 1, 7] The new challenge is to factor the compression process in a way that keeps the computational complexity in the initial encoding step, yet still allows the delivery step to precisely control the rate-distortion trade-off. In this paper, we classify these new compression formats as *QoS-adaptive* video. There are many possible uses for QoS-adaptive video.

QoS-adaptive video may be used to expand the life-expectancy of encoded content. For example, a movie may be encoded with highest quality possible, but deployed at low quality level this year, and re-deployed at higher quality next year when network capacity is greater or cheaper. QoS-adaptive compression has benefit because encoding is time and cost intensive.

QoS-adaptive video may be used to increase utilization on broadcast transmission channels. Transmission channels may multiplex VBR video flows to take advantage of statistical multiplexing. The utilization in such aggregate flows can be increased if video degrades gracefully during transient overload conditions. For video-on-demand (VOD) systems, this translates into higher numbers of simultaneous streams per unit cost.

Finally, QoS-adaptive video can be used for delivery over the current Internet, even with its best-effort service model. In particular, the thesis of this paper concerns how to best take advantage of QoS-adaptive video for streamed delivery over the Internet. Our initial focus is on VOD applications, where content is compressed offline and at delivery time is streamed from a storage server. Our work straddles the two areas of QoS-adaptive compression and network streaming.

The reader might wonder about what distinguishes streaming from download. For a video, we define the download model to be where the transfer of the video must complete before the video is viewed. Transfer and viewing are temporally sequential. With this definition, it is a simple matter to employ QoS-adaptive video. One algorithm would be to deliver the entire video in the order from low to high quality components. The user may terminate the download early, and the incomplete video will automatically have as high quality as was possible[1]. Thus, QoS-adaptive download can be implemented in an entirely best-effort, time-insensitive, fashion. On the other hand, we define the streaming model to be where the user views the video at the same time that the transfer occurs. Transfer and viewing are concurrent. There are timeliness requirements inherent in this definition, which can only be reconciled with best-effort delivery by a time-sensitive adaptive approach. The challenge we pose is this: at what level should this adaptation occur?

1.1 Anti-TCP Dogma

Numerous works on streaming video have asserted that TCP is undesirable for video and audio streaming, yet propose alternate solutions compatible with the same best-effort IP infrastructure[4, 11, 18, 15]. We identify two common objections at the root of this dogma.

1.1.1 Packet Retransmissions

One objection states that TCP's use of packet retransmissions introduces unacceptable end-to-end latency. The claim is that resending the data is not appropriate because, given the real-time nature of video, the resent data would arrive at the receiver too late for display. This latency constraint can be addressed through client-side buffer management—if it does not conflict with application-level latency requirements. The point of conflict occurs when the latency requirements are tight relative to path round-trip time (RTT), because a TCP sender's earliest detection of lost packets occurs in response to duplicate ACKs from the receiver. For interactive applications such as tele-conferencing or distributed gaming, users are highly sensitive to end-to-end delays more than 200 milliseconds. This end-to-end delay requirement persists for the duration of interactive applications. Our target application is VOD from stored media. Interactive events, such a start, pause, fast-forward, etc. are infrequent. Once the video is started, the user is not affected by end-to-end latency, as long as the video appears to play correctly.

An alternate problem with retransmission is its potential to limit the effectiveness of end-to-end feedback mechanisms. Namely the *QoS-feedback*, which controls the video's rate-distortion trade-off. We expect this control should operate on a much coarser time scale than RTTs, since frequent quality changes are unpleasant for the viewer. This implies that large client-side buffers are desirable, which is consistent with the results presented in [5, 16], where client-side buffers are large enough that QoS-adaptation can occur on timescales of minutes. With today's low costs for storage, it is feasible to buffer such large amounts of data.

1.1.2 Congestion Avoidance

Another objection states that abrupt rate variations due to congestion avoidance impede effective streaming. The congestion avoidance algorithms of TCP have been heavily studied and frequently discussed in the literature[6, 9, 13]. Briefly, the congestion algorithm is designed to probe available bandwidth, through deliberate manipulation of the transmission rate. The client application can smooth out the rate variations by employing buffering, which essentially borrows some current bandwidth to protect against future reductions. This only works for transient rate decreases, whose effects are small relative to the buffer capacity. There can and will be

[1]Wouldn't this be nice for Napster?

sustained reductions in TCP's rate, which will lead to buffer underflows regardless of buffer size. QoS-adaptation feedback mitigates sustained rate changes by adapting the rate-distortion trade-off.

Many TCP-friendly protocols with claims of better suitability for video have been proposed[4, 11, 18, 15, 19]. These protocols recognize the need for congestion avoidance, but propose mechanisms which provide service with smoother short-term rate behavior. Smoother transmission would allow a QoS-adaptive control to work with smaller client side buffers. Again though, we expect that our target application works better with large client side buffers because they afford less frequent QoS-adaptation. Furthermore, it is difficult in general to achieve acceptance and deployment of new network protocols[21].

1.2 QoS Adaptation: Application or System Level?

Some of the recent work on QoS-adaptive streaming has been based on tight integration between the QoS-feedback and the congestion avoidance mechanism[16]. In particular, the QoS-adaptation algorithm expects direct access to congestion avoidance state such as instantaneous round-trip time, and expects synchronization with congestion window adjustments. One way to achieve this is to implement the QoS-adaptation directly in the kernel, alongside the transport protocol. Alternatively, the transport interface to the user-level might be expanded to expose congestion avoidance behavior[3], but with this approach the precision of user-level control would be less, due the non-realtime performance provided by general purpose kernels. Either way, the advantage of tight integration with congestion control is similar to the payoff for smooth-rate transport; that is, a more responsive feedback mechanism which affords smaller client side buffers.

We think it is clear that a pure application level approach should be studied. At the very least, the performance of an application level solution is needed to serve as a control data to quantify the incremental benefit of system support for QoS adaptation.

The rest of this paper maps out our approach to video streaming. Section 2 describes our work at the QoS-adaptive video level. Section 3 discusses streaming over TCP. We compare our approach with related work in Section 4. Section 5 contains our concluding discussion.

2 QoS Adaptive Video

We describe our work on QoS-adaptive video in order to provide a foundation for introducing our approach to QoS-adaptive TCP-streaming. In QoS-adaptive video-on-demand (VOD), there is an intimate mutual dependency between video format and network transport, because rate-distortion adjustments as part of the streaming. We started our work at the video level, and assumed our target network would at least support the simplest model for QoS-adaptive delivery we could imagine: priority-labeled packets. We describe the video layer of our architecture now. In section 3, we'll discuss how we're using priority-drop with TCP streaming.

In addition to testing our choice of priority-drop discipline, we have another important goal. In our prior experience with QoS adaptation for video[20], we observed that video quality is multi-dimensional. For example, video has both temporal and spatial quality dimensions. We realized that adaptation policies were needed to specify the most appropriate use of available bandwidth[17]. In particular we wished the architecture to support deploying the same content with multiple adaptation strategies. The adaptation strategies could be used to tailor the content differently between users with conflicting requirements. For example, a user with a small screen may place less importance on spatial detail. For a different example, a user with limited processing might benefit more by dropping temporal quality[2]. We were therefore interested to see if a simple mechanism like priority-drop would be expressive enough to implement multi-dimensional QoS-adaptation strategies.

We describe our solution in three brief steps: our QoS-adaptive video format is described in section 2.1; section 2.2 will describe how adaptation strategies are specified and how a specification is mapped into a priority labeling for our QoS-adaptive format. Section 2.3 will show performance highlights of our approach based on our prototype implementation. Further details of our work on QoS adaptive video are available in [10].

2.1 SPEG: A spatial scalability extension for MPEG-1

Although layered scalability extensions are present in several of the common video compression standards, freely available implementations are not available. For our purposes, the easiest solution was to develop a rapid prototype with an open-source MPEG-1 software codec [14]. We call our modified MPEG-1 format SPEG[10]. SPEG adds spatial scalability to MPEG-1, through layered quantization of DCT data. Our current implementation has four levels of SNR scalability in each MPEG-1 picture. Our implementation does transcoding from MPEG-1 to add the spatial scalability, and fragments the stream so that spatial layers for each picture are in separate packets. Transcoding was convenient because it allowed us to reuse a maximum amount of available code and video content. The important design principle in SPEG is that a QoS-adaptive video format must fragment the video so that data contributing to orthogonal quality-dimensions are separated into distinct packets. This principle is the key to separating QoS-adaptation mechanism from policy.

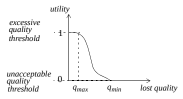

Figure 1: A utility function. The horizontal axis describes an objective measure of video quality, while the vertical axis describes the subjective utility of a presentation at each quality level. The region between the q_{max} and q_{min} thresholds is where a presentation is acceptable. The q_{max} threshold marks the point where lost quality is so small that the presentation is "as good as perfect." The area to the left of this threshold, even if technically feasible, brings no additional perceivable value. The rightmost threshold q_{min} delimits the point where lost quality has exceeded what is tolerable, and the presentation is no longer of any use. The utility levels on the vertical axis are normalized so that zero and one correspond to the "useless" and "as good as perfect" thresholds. In the acceptable region of the presentation, the utility function should be continuous and monotonically decreasing, reflecting the notion that decreased quality should correspond to decreased utility.

2.2 QoS Specification and Mapping

A utility function is the simple and general means we use to specify QoS preferences for video. Figure 1 depicts the general form of a utility function.

[2]In our experience, frame-dropping is more effective for reducing CPU requirements than dropping spatial detail

We have developed an algorithm for dynamically mapping from utility functions to a priority assignment for the packets of a QoS-adaptive video. Our implementation of the *QoS-mapper* works for the two quality dimensions, temporal and spatial resolution, supported by the SPEG format. Separate utility functions are given for each dimension. The current implementation employs 16 priority levels. We expect finer-grained adaptation would be necessary in practice, but 16 priority levels were enough for us to be able to get a good idea for the potential performance of the approach. We summarize our results next.

2.3 QoS-Mapping Results

In this section, we present results of experiments to characterize the adaptation performance of our approach. The adaptation performance is measured with respect to both presentation QoS and resource QoS. The experiments were conducted for three adaptation policies, as specified by different sets of utility functions. The presentation QoS results show that our approach supports tailorable adaptation in multiple QoS dimensions. The resource QoS results demonstrate that the adaptation covers a wide range and is evenly distributed.

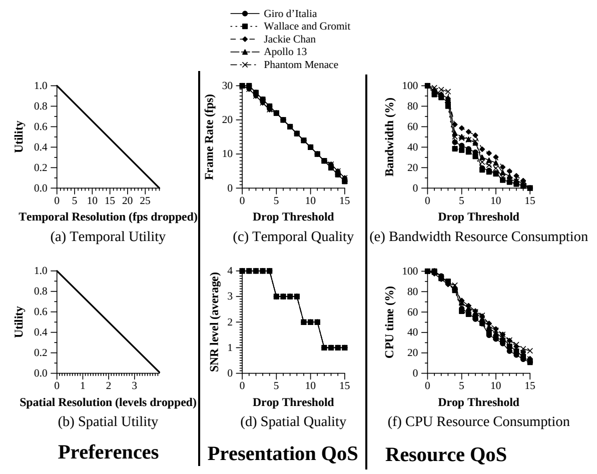

Figure 3: QoS Mapping Applied to SPEG

In figure 3(a) and (b) we see a QoS-adaptation strategy consisting of equal linear utility functions for temporal and spatial quality. We will explain the results for this strategy here. Further results for other strategies are presented in [10].

Figures 3(c) and (d), show the presentation QoS derived from this policy for various priority-drop thresholds. That is, given the priority-labeling produced by the QoS mapper, the graph shows what quality level is realized at each of the 16 priority-drop thresholds. The priority drop thresholds are on the horizontal axes of the graphs. An increased priority drop threshold means more packets are dropped.

Ideally, the presentation-QoS graphs would look the same as the utility functions they were derived from. In particular, the range of acceptable presentation QoS would be covered, and the shape of adaptation would follow the shapes of the utility functions. Figure 3(c) shows the relationship between presentation-QoS for temporal resolution (frame rate) and priority-drop threshold. It should be noted that figure 3(c) contains lines for each of the test movies, but they overlap very closely because the mapper is able to label packets to follow the utility function policy closely. Although desirable, this result was not entirely expected because MPEG's inter-frame dependencies constrain the order in which frames can be dropped, and some GOP patterns are particularly poorly suited to frame dropping. On the spatial resolution side, in figure 3(d), we note that our current mapper drops resolution levels uniformly across all frames, resulting in a stair-shaped graph, since there are only 4 SNR levels in SPEG. In as much as the SPEG format allows, the presentation-QoS matches the specified user preferences.

Resource QoS adaptation profiles are shown in the third pair of graphs in Figures 3(e) and (f). We show the average bandwidth of the movies at each drop threshold, as a percentage of the bandwidth when no packets are dropped. Similarly, we show the CPU time required for client side processing of the video at each drop threshold. A good shape for these graphs would be smooth and linear over a wide range of resource levels. We see that bandwidth in Figure 3(e) does indeed range all the way down to only a few percent, although there is a rather sharp drop when the first SNR layer is dropped. CPU time in Figure 3(f) is very nice and smooth, although it does not cover as much range as bandwidth, and reaches a minimum of about 10 percent. We also note that the movies are closely clustered in their resource-QoS graphs, indicating that adaptation is independent from differences in encoders or encoder parameters.

2.4 The Price of Adaptation

We now describe some of the performance costs associated with dynamic quality adjustment. Figure 4 compares performance for MPEG and SPEG versions of movies at the same presentation QoS level.

Given the wide range of adaptation shown in the adaptation experiments, the relatively small bandwidth overhead of SPEG is encouraging, especially considering the simplicity of the approach used in SPEG. The CPU overhead is more severe, but we know there is great room for improvement. The choice of transcoding SPEG back to MPEG was convenient for constructing the experiment, but is an obvious major source of un-necessary overhead.

We also stress that although our video implementation is based on MPEG-1 video, the techniques are applicable to the other most popular open formats: MPEG-2, MPEG-4, and DV[8, 1, 2]. We believe our approach will apply especially well to the upcoming Fine Granularity Scalability extension to MPEG-4 visual standard[12].

3 TCP Streaming Approach

We now consider the problem of streaming QoS-adaptive video over TCP. The main challenge is to do our best to take advantage of available bandwidth while maintaining reasonably smooth presentation quality. We use the notion of *goodput* to quantify the bandwidth efficiency. We define goodput as the rate of data arriving *on-time* at the client, thus contributing to the video quality experienced by the user. Some data will arrive late, reducing goodput. Since goodput is limited to the actual throughput provided by TCP, we should ensure that we maximize TCP's throughput. Given TCP's flow and congestion avoidance mechanisms[9], the way for an application to maximize the throughput of a TCP flow is to be *work-conserving* at the server—the server should always send data as fast as TCP will accept.

With our QoS-adaptive video architecture, goodput translates into two-dimensions at the application level. One dimension of goodput is the progress in time. The other dimension of goodput is the quality-level, as embodied in packet priorities. Keeping these two dimensions in mind will help in understanding our approach to streaming. They are central notions in our streaming model, which we now describe.

Video	Resolution	Length (frames)	GOP pattern
Giro d'Italia	352x240	1260	BBIBBPBBPBBPBBP
Wallice and Grommit	240x176	756	IPI
Jackie Chan	720x480	2437	IBBBPBBB
Apollo 13	720x480	864	BBIBBP
Phantom Menace	352x240	4416	BIBPBPBPBPBPBPBP

Figure 2: Movie Inputs. The movies were coded with several different MPEG encoders. A variety of content types, movie resolutions, and GOP patterns were chosen to verify our techniques perform consistently.

Video	MPEG bandwidth (Mbps)	SPEG bandwidth (Mbps)	Increase bandwidth (%)	MPEG CPU (secs)	SPEG CPU (secs)	Increase CPU (%)
Giro d'Italia	1.823	2.121	16.3	45.8	72.9	59
Wallice and Grommit	0.968	1.081	12.7	12.1	16.6	37
Jackie Chan	1.839	2.479	34.8	216	252.2	17
Apollo 13	3.474	4.193	20.7	89.3	121.5	36
Phantom Menace	1.228	1.313	6.9	103.7	180.4	74

Figure 4: Overhead of SPEG

3.1 Priority-Progress Streaming Model

The basic abstraction we use for QoS-adaptive video is what we call a *priority-progress stream*. A priority-progress stream is a sequence of packets, each with a timestamp and a priority. The timestamps expose the timeliness requirements of the stream, and allow progress to be monitored. The priorities allow informed-dropping in times of resource overload.

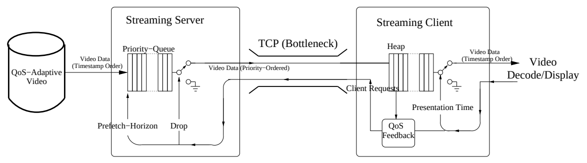

Figure 5: Priority-Progress Streaming Streaming

The priority-progress streaming model is depicted in figure 5. A server and client process are separated by a bottleneck. The bottleneck provides best-effort, reliable service. TCP is the bottleneck we're concerned with in this paper. The server sends the packets through the bottleneck to the client at a rate which matches the progress rate with the real-time rate of the presentation. If the bottleneck can't keep up with the sender's current data rate, the sender reduces the rate through priority-dropping. The data-rate of the stream will often be kept somewhat below the rate of the bottleneck in order to allow client side buffers to reach some target fill level. The data held in buffers will be used to absorb transient reductions in the bottleneck rate. Before we consider the details of the streaming algorithm and protocols, we must refine some details of packet semantics.

We assume no two packets may be labeled with the same pair of timestamp and priority. This is a logical consequence of our informed-dropping objective. Since we wish all dropping decisions to be informed, it does not make sense to have packets that can not be distinguished, either by priority or timestamp. If we did, the dropping algorithm might be faced with making arbitrary decisions: given a pair of packets with equal timestamp and priority, which is the better to drop? If there really are such arbitrary relationships in the data, we disambiguate such packets ahead of time in the QoS-mapper stage.

Each packet in a priority-progress stream also contains one additional label, a *dependency-barrier* flag. In our model, the priority ordering is *cumulative*, lower priority data always depends on at least some of the higher priority data. This reflects the structure of quality-adaptive video formats, which use cumulative quality layers. The value of the dependency barrier is true when all preceding packets are free of dependencies on the current or any subsequent packets. The dependency-barrier flag is meant to capture the *scope* of cumulative dependencies, information which the streaming algorithms can use to improve goodput. Without the flag, an adaptive dropping mechanism might never be sure that it is not sending a packet which has a dangling-dependency. A packet has a dangling dependency when some other packet on which it depends is dropped. Whenever the server's sending rate is increased, priorities alone provide insufficient information to avoid sending dangling-dependency packets. Sending such dangling-dependency packets is a waste of throughput, and it eats away from potential goodput. The streaming algorithm can avoid this problem by waiting for the next dependency barrier before it raises the sending rate.

This completes our model of QoS-adaptive streaming. To summarize, each packet in a priority-progress stream contains three labels: time-stamp, priority, and dependency barrier.

3.1.1 The Streaming Algorithm: Priority-Dropping and Client-Buffer Management

Our QoS-adaptive streaming protocol is client-driven. The server periodically receives requests from the client which consist of two values: a drop time, and a prefetch-horizon time (see Figures 5 and 6). In response to the client-messages, the server pushes data in a best-effort fashion. The times in the client request are in the same time units as the priority-progress packet timestamps. At any given time, the server is sending packets with timestamps between the current drop time and prefetch-horizon time, *in priority-order*. As we mentioned earlier, the server will attempt to send as fast as TCP will allow. The client side of the algorithm regularly send request messages to the server to advance the two times forward. The server will not normally reach the prefetch-horizon time before the times are advanced by the client[3].

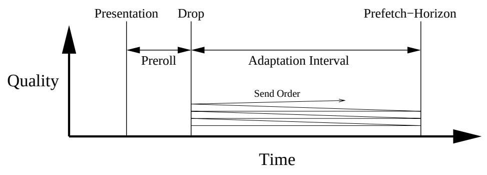

Figure 6: QoS-adaptation algorithm timeline

Each time the server receives a new request from the client, some low priority packets will be dropped and high-priority packets become newly eligible for sending. The dropped packets are those whose timestamps are less than the new drop-time. The number of packets dropped will directly reflect the deficit between the available bandwidth and the maximum bandwidth requirements of the stream. In this way, the server automatically discovers the correct packets to send. The newly eligible packets are those within the new prefetch-horizon time. The server uses an enhanced form of priority-queue to reorder the packets by priority. The server also maintains a separate queue to track of dependency barriers in the stream, and adjusts the drop-time and prefetch-horizon to align with the barriers. This ensures dangling-dependencies do not occur at the client.

The client side of the algorithm uses heap data structure to reorder data back into timestamp order before passing it downstream to the video decoder. The client advances the drop-time at the same rate as the presentation rate, although it computes the drop time by adding a constant *preroll* offset to the presentation time. The preroll should be set to a conservative estimate of the worst case latency for client-requests to reach the server. The client can control the balance between stability and responsiveness of the QoS-adaptation by adjusting the size of the *adaptation interval*, which is the distance between the drop time and the prefetch-horizon time.

We are currently implementing these algorithms and setting up tests to measure there performance through TCP simulations, as well as real-world tests with our existing QoS-adaptive video player implementation.

4 Related Work

Our approach is distinct from similar work on TCP-friendly quality adaptive streaming [16] in that our adaptation algorithm operates at the application level. We do not require detailed information about the congestion avoidance decisions in the protocol stack. We do not believe such a tight integration is necessary for Internet video-on-demand, and favor avoiding unnecessary changes to operating-system kernels. Feng et al. have had good results with priority-based technique for streaming[5]. Our approach extends theirs to a richer priority-progress model, which offers the added capability of multi-dimensional adaptation and tailorable adaptation policies.

5 Discussion

We have argued the case for streaming video with TCP. We claim that TCP is a viable and effective choice. We support this claim with our experience in developing a QoS-adaptive video architecture, and describe the design of the TCP-streaming system we are implementing. TCP has strong pragmatic advantages because it is ubiquitous. Alternatives to TCP must have clear advantages if they are to justify the significant hurdles of acceptance and deployment in today's Internet.

Our current plans are to measure the effectiveness of our streaming approach using a mixture of results driven by TCP simulation and real-world measurements with our existing QoS-adaptive video implementation. We are also planning to explore the effectiveness of our QoS-adaptation approach for addressing the case where client CPU is the bottleneck. The best-effort service model provided by TCP is analogous to the behavior of software video decoders, especially in the case of variable-bitrate compressed videos.

[3]If it does, it means the network has enough available bandwidth to handle the video at maximum quality.

References

[1] ISO/IEC 14496-2. Information technology — coding of audio-visual objects — part 2: Visual. International Standard, December 1999. First edition.

[2] IEC 61834. Helical-scan digital video cassette recording system using 6,35 mm magnetic tape for consumer use (525-60, 625-50, 1125-60 and 1250-50 systems). International Standard.

[3] Hari Balakrishnan, Hariharan Rahul, , and Srinivasan Seshan. An integrated congestion management architecture for internet hosts. In *Proceedings of ACM SIGCOMM '99 Conference*, Cambridge, MA, October 199.

[4] Shanwei Cen and Jonathan Walpole. Flow and congestion control for internet streaming applications. In *Proceedings Multimedia Computing and Networking (MMCN98)*, 1998.

[5] Wu chi Feng, Ming Liu, Brijesh Krishnaswami, and Arvin Prabhudev. A priority-based technique for the best-effort delivery of stored video. In *SPIE/IS&T Multimedia Computing and Networking 1999*, San Jose, California, January 1999.

[6] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17, 1989.

[7] G. Cote, B. Erol, M. Gallant, and F. Kossentini. H.263+: Video coding at low bit rates. *IEEE Transactions on Circuit and systems for Video Technology*, 8(6), November 1998.

[8] Barry G. Haskell, Atul Puri, and Arun N. Netravali. *Digital Video: An Introduction to MPEG-2*, chapter 9. Chapman & Hall, 1997.

[9] Van Jacobson and Michael J. Karels. Congestion avoidance and control. In *In Proceedings of ACM SIGCOMM'88*, pages pp. 79–88, August 1988.

[10] Charles Kasic and Jonathon Walpole. QoS scalability for streamed media delivery. CSE Technical Report CSE-99-011, Oregon Graduate Institute, September 1999.

[11] J.R. Li, D. Dwyer, and V. Bharghavan. A transport protocol for heterogeneous packet flows. In *IEEE Infocom'99*, 1999.

[12] Weiping Li, Fan Ling, and Xuemin Chen. Fine granularity scalability in mpeg-4 for streaming video. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, Geneva, Switzerland, May 2000. IEEE.

[13] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput: A simple model and its empirical validation. In *In Proceedings of ACM SICOMM'98*, 1998.

[14] K. Patel, B. C. Smith, and L. A. Rowe. Performance of a software mpeg video decoder. In *Proceedings ACM Multimedia 93*, pages pp. 75–82, Anaheim, CA, August 1993.

[15] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestiong control mechanism for realtime streams in the internet. In *Proceeedings of IEEE Infocomm*, March 1999.

[16] Reza Rejaie, Mark Handley, and Deborah Estrin. Quality adaptation for congestion controlled video playback over the internet. In *Proceedings of ACM SIGCOMM '99 Conference*, Cambridge, MA, October 1999.

[17] Richard Staehli, Jonathan Walpole, and David Maier. Quality of Service Specification for Multimedia Presentations. *Multimedia Systems*, 3(5/6), November 1995.

[18] Wai tan Tan and Avideh Zakhor. Internet video using error resilient scalable compression and cooperative transport prototocl. In *Proc. ICIP*, volume 1, pages 17–20, 1998.

[19] The TCP-friendly website. http://www.psc.edu/networking/tcp_friendly.html.

[20] Jonathan Walpole, Rainer Koster, Shanwei Cen, Crispin Cowan, David Maier, and Dylan McNamee. A player for adaptive MPEG video streaming over the Internet. In *Proceedings 26th Applied Imagery Patter Recognition Workshop AIPR-97*, Washington, DC, October 1997. SPIE.

[21] David Wetherall, Ulana Legedza, and John Guttag. Introducing new internet services: Why and how. *IEEE Network Magazine*, July/August 1998.