Quality-Adaptive Media Streaming by Priority Drop*

Charles Krasic, Jonathan Walpole, Wu-chi Feng
OGI/OHSU
Beaverton, Oregon
krasic,walpole,wuchi@cse.ogi.edu

ABSTRACT

This paper presents a general design strategy for streaming media applications in best effort computing and networking environments. Our target application is video on demand using personal computers and the Internet. In this scenario, where resource reservations and admission control mechanisms are not generally available, effective streaming must be able to adapt in a responsive and graceful manner. The design strategy we propose is based on a single simple idea, priority data dropping, or *priority drop* for short. We evaluate the efficacy of priority drop as an adaptation tool in the video and networking domains. Our technical contribution with respect to video is to show how to express adaptation policies and how to do *priority-mapping*, an automatic translation from adaptation policies to priority assignments on the basic units of video. For the networking domain, we present *priority-progress* streaming, a real-time best-effort streaming protocol. We have implemented and released a prototype video streaming system that incorporates priority-drop video, priority mapping, and priority-progress streaming. Our system demonstrates a simple *encode once, stream anywhere* model where a single video source can be streamed across a wide range of network bandwidths, on networks saturated with competing traffic, all the while maintaining real-time performance and gracefully adapting quality.

Categories and Subject Descriptors: C.2.2 [Computer Systems Oraganization]: Network Protocols

General Terms: Algorithms, Measurement, Experimentation

Keywords: Quality Adaptive Streaming, Priority Mapping, Internet

*This work was partially supported by DARPA/ITO under the Information Technology Expeditions, Ubiquitous Computing, Quorum, and PCES programs and by Intel.

NOSSDAV'03, June 1–3, 2003, Monterey, California, USA.

1. INTRODUCTION

The Internet is a best-effort environment, where users come and go without resource reservations or admission control. The Internet relies on a model of voluntary cooperative sharing, the foundation of which is congestion control in transport protocols, mainly TCP. It is widely acknowledged that the use of congestion control has been an essential part of the Internet's stability to date [7]. If streaming traffic is to avoid threatening the overall stability, then it too must employ congestion control. Congestion control adapts the sending rate of a flow to share with other flows on the path, changing rates as traffic from other flows comes and goes. As a result, Internet traffic is very bursty. Numerous studies of Internet traffic patterns have shown that traffic rates exhibit significant variation over the full range of time scales, exhibiting so-called self-similar behavior [5, 30]. Given this burstiness, it follows that it is highly unlikely that a single target bitrate will suffice for Internet streaming. If the rate estimate is too conservative, the video stream will under utilize the network, and the resulting video quality will be lower than necessary. On the other hand, if the rate estimate is too aggressive, then the transfer can not complete in real-time and so there will be a streaming failure. For longer duration streaming, the chances are good that a single rate will be too conservative at some times *and* too aggressive at others. Several researchers have recognized these issues, and proposed quality-adaptation instead of single-rate streaming [31].

There have been many proposed techniques for the adaptive delivery of compressed video data over networks. The common idea of quality-adaptive streaming techniques is to adapt dynamically to environmental changes through adjustments in the rate-distortion ratio of the video. Adapting quality over best-effort networks is extremely difficult given the bit-rate changes that occur over time in both.

In this paper, we describe a design strategy for quality-adaptive streaming software. Our strategy revolves around the idea of using priority data dropping, *priority drop* for short, as the primary means of adaptation. In priority drop, the basic data units of the media are explicitly exposed and appropriately prioritized, with the goal that priority-order dropping of data units will yield a graceful reduction in media quality, as we will show in the experimentation section. Our contributions are in two areas, video adaptation and network streaming.

The video component of our system makes compressed video streaming friendly through support of priority drop. We describe a video format, called SPEG (Scalable MPEG),

to illustrate how current video compression techniques can be extended to support priority drop[1]. In contrast to random dropping, which results in unusable video at dropping levels of just a few percent, priority drop is informed and can achieve graceful degradation, over more than an order of magnitude in rate. One of the main questions that arises when considering such a range of target rates is what aspect or aspects of video to degrade? The answer can be influenced by several factors, such as the nature of the content, the nature of the viewing device, the personal preferences of authors, viewers, etc. For example, a sports program might benefit most from preserving fidelity of motion, perhaps at the expense of color fidelity. A user with a PDA may place low relative importance on spatial resolution, compared to a user with a full sized screen.

The scalable coding aspects of SPEG are not our main focus, but rather our main contribution to video entails efficient support for *tailorable* adaptation. We describe a simple method to specify adaptation policies and an associated *priority-mapping* algorithm. The priority-mapping algorithm translates the policy specifications into appropriate priority assignments on data units of priority-drop video. With this approach, video compression is decoupled from the final adaptation, which opens the possibility that each piece of content may be adapted in different ways for different scenarios, with far lower complexity than actually recompressing or transcoding the content during streaming. This added flexibility makes content more re-usable. The ideal streaming-friendly media format would have the characteristics of "encode-once, stream anywhere."

The second component of our contribution is in network streaming. We present an algorithm for real-time best-effort streaming called Priority-Progress streaming (PPS). PPS combines *data re-ordering* and *dropping* to maintain timeliness of streaming in the face of unpredictable throughput. The data units of the priority-drop video are sent in priority order. The algorithm is best effort in that it allows the congestion control mechanism to decide appropriate sending rates. When this sending rate is low, the timeliness of the stream is maintained by dropping low-priority data units at the sender, *before* they would otherwise reach the network. In this way, the amount of higher-priority data sent automatically matches the rate decisions of the congestion-control mechanism.

We have implemented a streaming system which integrates SPEG, priority-mapping, and PPS, with the following results. First and foremost, it maintains timeliness of the stream in the face of rate fluctuations in the network. Second, PPS makes full use of available bandwidth, and achieves full goodput from that bandwidth, thereby maximizing the average video quality. The bandwidth used is limited by the congestion control of the underlying transport, in our case TCP, so the usage represents a fair share, in friendly consideration of the Internet's existing traffic mix [7]. Third, it starts quickly when the user initiates the stream, avoiding a long pre-buffering period. Finally, it limits the number of quality changes that occur, by using bandwidth skimming to increase client-side buffering concurrent during normal playout. The overall message of our results is that priority drop is very effective: a single video can be streamed across a wide range of network bandwidths, on networks heavily saturated with competing traffic, while maintaining real-time performance and gracefully adapting quality.

The remainder of this paper is organized as follows. In the next section we discuss problem background and related work. In Section 3, we describe priority-drop video and priority mapping. In Section 4 we describe the details of the PPS algorithm. Section 5 presents experiments and results from our prototype streaming video system. Finally, discussion and conclusions are in Section 6.

2. BACKGROUND

Most streaming content on the Internet today is provided using one of three streaming platforms: Microsoft's Windows Media, Real Networks RealSystem, and Apple's QuickTime. To various degrees, these systems adhere to a suite of standards related to streaming such as RTP, RSTP, SIP, and SMiL. [9, 8, 26, 27]. Quality adaptation algorithms are outside the scope of any of these standards. In particular, while adaptation might be layered on RTP, RTP does not provide any direct algorithm for quality adaptation. These commercial systems all employ proprietary quality adaptation. Although these proprietary adaptation mechanisms are largely secret, we make some high level observations in the following paragraphs based on published information.

In addition to commercial activity, there has been extensive academic research related to video streaming over the Internet. The research spans several distinct domains, including video compression, real-time systems, and networking. A spectrum of adaptive strategies have been proposed to deal with the consequences of best effort service [31]. One of the most commonly used strategies is *one-time adaptation*, where the user chooses between a small set of predetermined rates before streaming begins. Once started, streaming is fixed at this single-rate regardless of competing traffic, hence this approach retains the basic problems of single-rate streaming mentioned earlier, where it is prone to yield lower quality than necessary when more bandwidth is available and prone to complete failure when less bandwidth is available. Both problems are more probable for longer duration content. Apple's Quicktime uses one-time adaptation and in addition it adjusts the amount of client-side buffering based on measured rate volatility during startup [29]. Startup time, while initial buffering is established, can be quite high—on the order of tens of seconds. Windows Media and RealSystem based systems are often configured in this mode also, even though they do support more advanced mechanisms, which we'll describe below. In the remainder of this section, we expand on the basic performance issues for quality-adaptive streaming, in light of some of the approaches proposed in the literature and in terms of the commercial streaming systems.

The related work to this paper falls into four main categories. *Multi-version* techniques store a single video at a range of pre-selected bitrates (e.g. Windows Media IntelliStream and Real's SureStream) [4, 1]. While simple to implement, multi-version supports only coarse adaptation and under utilizes storage. *Online scaling* techniques support changing the target rate parameter of the encoder or the transcoder on the fly. While these support fine-grained adaptation, the computational time required to recode limits scalability of these approaches. *Scalable video coding* technologies focus on creating compression formats that allow

[1]SPEG is similar to MPEG FGS, but easier to implement with publicly available software

adaptation of the rate-distortion relationship without explicitly re-coding (e.g. MPEG-2 scalability, MPEG-4 FGS). These techniques are complementary to the work we describe here. Advances in these areas can be directly incorporated into our framework. While the first three categories are concerned mainly with video representation and coding, the fourth category is *adaptive streaming* which concerns the mechanics of actual network delivery.

Ideally, a quality-adaptive streaming system will select video quality to match the average available network bandwidth. In practice, adaptation tends to be limited to discrete steps, and consequently the rate match is only approximate. A system that supports steps with finer-granularity generally results in a better match, which manifests itself in higher quality and better reliability of streaming. The type of video compression, especially whether the compression is scalable or not, is a major factor influencing the granularity of quality-adaptive streaming.

Because many of the compression formats in common use are not explicitly scalable [15, 14, 16, 13], the target rate is a required parameter for encoding. These formats do not provide explicit support for adapting rate after encoding. Frame dropping is a well known work-around, and is probably the most popular video adaptation mechanism, having been used since the first quality-adaptive Internet streaming systems appeared [2].

Online-scaling techniques, which include live encoding, transcoding, and data-rate shaping (DRS), allow changing the target rate parameter of the encoder or transcoder on the fly [17, 32]. Transcoding and DRS can have significantly lower computational complexity than encoding. The main advantage of online scaling is very fine granularity. However, even the most efficient DRS is very computationally intensive relative to non-adaptive streaming, or adaptive streaming through frame dropping or multi-coding. This extra computational cost poses a major obstacle to supporting very large numbers of independently adaptable streams in servers and edge devices.

In contrast, scalable compression aims to support low-complexity adaptation that will scale to large numbers of streams. Scalable compression schemes explicitly support multiple quality levels, exposing two or more layers in the encoded video. The layers are progressive, the higher layers depend on the lower layers, and the higher layers are used to refine quality. The various scalable compression approaches differ in terms of granularity, ranging from very coarse, as in the work in Layered Multicast [23] and MPEG-2 Scalability [10], to very fine, such as in recent work in MPEG-4 and H.26L Fine Granularity Scalability [22, 11]. With the current state of the art, scalable video compression comes with a compression efficiency penalty, in that video quality is lower compared to the results of non-scalable compression at the same rate, but this penalty is getting smaller [11]. Fine granularity scalability through layering makes it possible to begin streaming without even knowing the target rate, by sending lower layers before higher layers and truncating higher layers if time runs out. Contrast this approach with online-scaling, where the quality adaptation must commit to a target rate *before* encoded data is ready to transmit. In exchange for the small efficiency penalty, scalable compression offers a significant boost in freedom for the design of adaptive streaming mechanisms.

A principal concern with streaming is the potential impact of video traffic on existing Internet traffic. Many research projects have studied quality adaptive streaming in relationship with *TCP-friendly* congestion control [31, 25, 3, 6, 17, 28, 19]. A common idea among them is to let the transport protocol and its congestion control dictate the appropriate sending rate. The main differences are in the details of deciding what to send and what to drop, and what information are used to inform these control decisions. For example, Rejaie *et al* describe their algorithms for optimal streaming [25], where optimal means minimal client-side buffering, and thus a minimal associated contribution to end-to-end latency. The role of their algorithm is to control adding and removing quality layers, where the control decisions are based on a rate-driven feedback control. The design of their control is based on analysis of additive-increase multiplicative-decrease (AIMD) congestion control[2] and an assumption of apriori knowledge of video rate requirements [25]. *Feamster et al* extend this work to more general congestion control mechanisms [6]. In contrast to these systems that explicitly attempt to match rates, *Feng et al* describe an adaptive streaming algorithm that uses a sliding window over video frames, sending data from low to high quality, in best effort fashion [3]. Feng's algorithm gains simplicity because it does not attempt to absolutely minimize client-side buffering, and has the advantage of working without direct assumptions about the design of the underlying congestion control. Kang et al. [18] propose a priority-driven adaptation, but assuming fixed bandwidth channels. The question of how to link scalable video encoding and tailorable adaptation policies to TCP-friendly streaming is open, and is the topic of this paper.

We use scalable compression and TCP in this paper. One of the contributions of our approach is to demonstrate the benefits of using the priority-timestamp packet as the basic unit of media abstraction, as opposed to video frames, or layers in a stream. Through priority-mapping, we extend scalable video compression to support tailorable adaptation, so that compromises made in quality better reflect the influence of specific content, viewing devices, and user preferences. Our Priority-Progress Streaming algorithm extends TCP-friendly adaptive streaming to support direct control over quality compromises in streaming, such as latency limits, and limits on the number of quality changes, while preserving the goals of high utilization and video quality.

3. STREAMING-FRIENDLY VIDEO

In this section we will describe how scalable video compression can support tailorable adaptation through priority drop. This consists of a scalable video format and a *Priority Mapper*. We have implemented an adaptive streaming system based on our approach, called the Quasar Video Pipeline. In lieu of a freely available implementation of the more recent scalable compression systems [22, 11], we have developed a minimal scalable compression format we call SPEG (Scalable MPEG), derived from MPEG-1 video. Our purpose in implementing SPEG was to test priority mapping and PPS using real video. Priority mapping is the main subject of this section, but we first give a brief description of SPEG for the benefit readers not familiar with scalable compression formats such as MPEG-4 FGS.

[2]TCP's congestion control uses an instance of AIMD after it reaches steady state.

3.1 Scalable Video

In MPEG video, each frame is broken down into 8x8 pixel blocks, which are converted to corresponding 8x8 blocks of coefficients using the discrete-cosine transform (DCT). *Quantization*, strategic removal of low order bits from these coefficients, is the primary basis for compression gains in MPEG and very many other similar compression schemes. SPEG transcodes MPEG coefficients to a set of levels, one base level and three enhancement levels as follows. If we denote the original MPEG coefficients $X[i,j]$, then SPEG partitions this coefficient data according to the following equations[3]:

$$\begin{aligned} X_{base}[i,j] &= X[i,j] >> 3 \\ X_{e0}[i,j] &= (X[i,j] >> 2) \;\&\; 1 \\ X_{e1}[i,j] &= (X[i,j] >> 1) \;\&\; 1 \\ X_{e2}[i,j] &= X[i,j] \;\&\; 1 \end{aligned}$$

The coefficients from each level are grouped to form layers, four per original MPEG frame, which are the basic *application level data units* (ADUs) in SPEG. The above steps can be reversed to return SPEG back to the original MPEG. Alternatively, we can drop some or or all of the enhancement layer ADUs (from high to low) substituting zero values for the missing data. The effect of such dropping is analogous to having used higher quantization parameters during MPEG encoding, yielding lower bitrate in exchange for less spatial fidelity. We present SPEG because it suffices to demonstrate the essential properties of scalable compression and because it is readily available to us. Our techniques would apply to most scalable formats, e.g. MPEG-4 FGS.

We expect future scalable codecs will expose even more scalability mechanisms. One example is *spatial-size scalability*, where the number of pixels of height and width are scalable. Another example is *chroma scalability* which might allow a range of color fidelities, from 4:4:4 to 4:2:2 to 4:1:1 to greyscale to monochrome. The object based compression techniques might allow *content adaptation* through addition and removal of objects[16]. These possibilities raise the issue of tailorable adaptation. In order to take full advantage of all of these scalability options, there would need to be a good way to control how they are used together. To explore tailorable adaptation, we use SPEG's spatial scalability in combination with frame dropping to provide a minimal example of a compression scheme with more than one scalability mechanism.

3.2 Priority Mapping

Having more than one quality dimension leads to the issue that choosing how to best adapt the multiple dimensions may depend on the usage scenario. For example, the target device may have a small screen, so preserving frame-rate may make more sense than spatial detail. A user may want to repeat a scene in slow motion, which looks smoother if more frames are inserted. Conversely, skipping frames is harder to notice when doing fast-forward scan. We have designed a priority-mapper with the intent of providing a general and flexible approach to tailoring quality adaptation to such specific quality preferences. The priority-mapper automatically assigns priorities to the units of a media stream, so that priority drop yields the most graceful degradation, as appropriate to the viewing scenario.

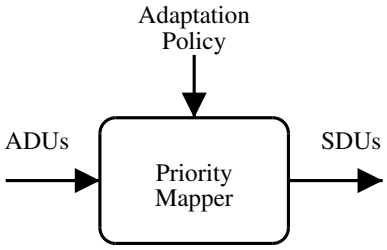


Figure 1: Priority Mapper

Figure 1 depicts the mapper used in the Quasar Video Pipeline. The mapper's inputs are *application data units (ADUs)* and the quality adaptation policy. The mapper's output are *streaming data units (SDUs)* which are aggregates of prioritized ADUs, where the aggregation is based on ADUs which have the same priority and timestamp value. The purpose of the aggregation is to isolate the PPS algorithm from low level details of the video format, particularly the data dependencies that exist between ADUs.

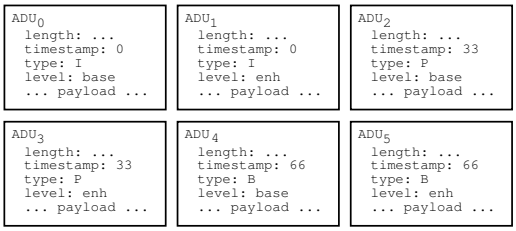


Figure 2: ADUs

Figure 2 shows a sequence of ADUs. The ADUs have a packet like form, consisting of a fixed-length header, and a variable length payload. The header contains basic information needed by the mapper, such as the length of the payload, a timestamp, and payload specific flags. For example, with SPEG these flags indicate the type of MPEG frame the ADU is part of (I, B, or P), and to which spatial scalability layer the ADU belongs[4].

3.2.1 Specification of Adaptation Policies

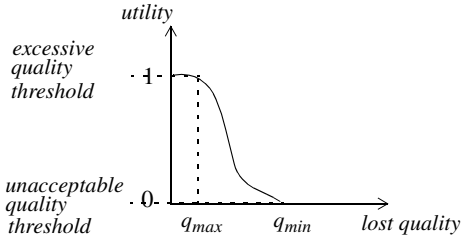


Figure 3: A utility function with thresholds

[3]The >> denotes the right bitwise shift operator, and the & denotes the bitwise *and* operation.

[4]To simplify our examples, figure 2 depicts only two spatial layers, although our SPEG implementation has four.

We use *utility functions* as declarative specifications for the adaptation policy. A utility function is a simple and general means for users to specify their preferences. Figure 3 depicts the general form of a utility function. The horizontal axis describes an objective measure of lost 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 user considers the presentation "as good as perfect." The area to the left of this threshold, even if technically feasible, brings no additional value to the user. The rightmost threshold q_{min} demarks the point where lost quality has exceeded what the user can tolerate, 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. In the case of priority mapping for SPEG, the adaptation policy consists of two utility functions, one for spatial quality and one for temporal quality.

3.2.2 Automatic Translation from Policy to Priorities

The mapping algorithm subdivides the timeline of the media stream into intervals called *mapping windows*. The size of the interval is a parameter to the mapping algorithm, but may be adjusted in order to meet alignment requirements; for example, the mapping window is a sequence of one or more complete GOPs for SPEG. The mapping algorithm prioritizes the ADUs within each window separately. For all ADUs in a given mapping window, the mapping algorithm finds the order in which ADUs may be dropped that has the minimum impact, given the data dependency rules of the video and the preferences specified via the given utility functions. The final priority assignment will be used by the streaming algorithm to guide quality adaptations, while accurately reflecting user preferences.

We use the ADUs from figure 2 as an example mapping window, which consists of a single GOP and spans the interval 0–66 ms. The priority mapping algorithm processes the ADUs within a window in two phases.

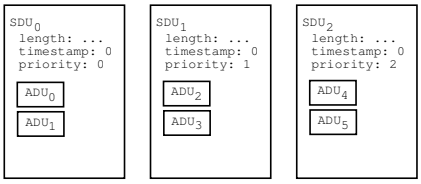
In the first phase, the ADUs are partially ordered, according to a *drop before* relationship[5], based on video data dependencies. For example, the spatial layering requires that base layer ADUs should not be dropped before their corresponding enhancement layer ADUs, which applies to the ADUs of figure 2 as follows:

$$ADU_1 => ADU_0 \quad ADU_3 => ADU_2 \quad ADU_5 => ADU_4$$

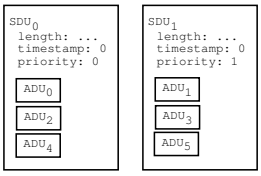
Similarly, MPEG's predictive coding rules (for I,P,B frames) are expressed as follows:

$$ADU_4 => ADU_2 => ADU_0$$

These first two sets of ordering constraints represent *hard dependency* rules, in that they simply reflect SPEG semantics. The mapper adds some other *soft dependency* rules which improve adaptation results. With video, for example, the mapper would add soft-dependencies so to ensure that frame dropping be as evenly spaced as possible[6]. After the first mapping phase, there still remains significant freedom for adaptation. For example, figure 4 contains two very different mappings for the ADUs of figure 2, yet both mappings adhere to the phase one constraints above.



(a) Frame drop



(b) Spatial drop

Figure 4: SDUs: prioritized and grouped ADUs

The second phase of the priority mapper algorithm is where the adaptation policy is used to refine the partial ordering from the first phase, generating the (totally ordered) prioritized SDUs.

The algorithm works through an iterative process of elimination over the ADUs. We say an ADU is *alive* if it is still in the set of unprioritized ADUs, and *dead* otherwise. Each iteration considers the set of candidate ADUs which are not yet dead, initially all ADUs from the mapping window, and have no living dependents, based on the constraints generated by the first phase. For each of these candidate ADUs, and each quality dimension (spatial and temporal in SPEG), the mapper computes the presentation quality would result if the candidate ADU were dropped, that is, the quality is computed based on all ADUs that are still alive, less the current candidate. For the temporal quality dimension, the mapper computes the frame rate, and for spatial quality the spatial level. At this point the mapper is ready to apply the adaptation policy. The utility functions are used directly to convert the computed quality values to corresponding utilities. The "overall utility" for each ADU is just the *minimum* of its per dimension utilities. The candidate ADU that has the highest utility is selected as the next victim; i.e. dropping this ADU next has the smallest impact on utility. The priority value for the victim ADU is a linear (inverse) fitting of the utility into the range of priority values. For example, in the Quasar pipeline this fit goes from a utility range of 0 to 1 to a priority range of 15 to 0[7]. The iterations stop when all ADUs have been assigned a priority.

Once all the ADUs have priorities, they are then grouped into SDUs, one per priority level. The SDUs are all set to

[5]This is really drop no-later than, since dropping is always optional.

[6]If half the frames are to be dropped, then it is best to drop every other frame, as opposed to more clustered dropping such as keeping even GOPs and dropping odd GOPs

[7]Maximum priority is 15

Video	Resolution	Length (frames)	GOP length
Giro d'Italia	352x240	1260	15
Wallice and Grommit	240x176	756	3
Jackie Chan	720x480	2437	8
Apollo 13	720x480	864	6
Phantom Menace	352x240	4416	16

Figure 5: 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.

have the timestamp of the first ADU in the window. This grouping simplifies matters for later stages, like the PPS algorithm and the video decoder[8].

3.3 Mapping Results

We now present some the results of mapping for several test movies. Figure 5 describes the set of movies used, which were prepared with a variety of encoders and encoder parameters. In figure 6(a) and (b) we set a quality adaptation policy consisting of equal linear utility functions for temporal and spatial quality. Figures 6(c) and (d), show the priority-assignment produced by the mapper. At each threshold, the quality corresponds to when all packets with priority lower than the threshold are dropped. For example, at priority threshold 6, 20 fps is achieved at SNR level 3.

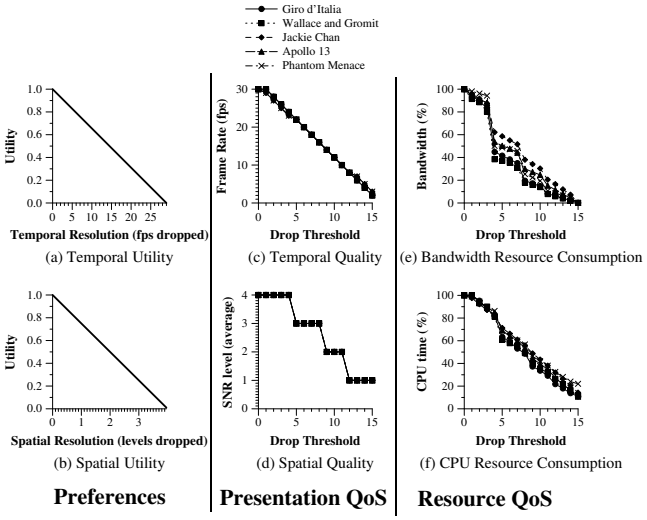


Figure 6: QoS Mapping Applied to SPEG

Ideally, the presentation quality 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 6(c) shows the relationship between presentation-QoS for temporal resolution (frame rate) and priority-drop threshold. It should be noted that figure 6(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 6(d), we note that the 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.

The resource side of the adaptation profiles are shown in the third pair of graphs in Figures 6(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 (decoding) of the video at each drop threshold, where the values are normalized to the CPU cost when no packets are dropped. A good shape for these graphs would be smooth and linear over a wide range of resource levels. We see that bandwidth in Figure 6(e) does indeed range all the way down to only a few percent. What this means is that the quality-mapper can prioritize the video to operate in extremely diverse networking and computing environments. CPU time in Figure 6(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. Further results for other policies are presented in [21].

4. PRIORITY PROGRESS STREAMING

In this section, we present an overview of the PPS algorithm. While the priority-drop video encoding and the priority-mapper described in the previous section do a substantial amount of preparation for delivery, the streaming algorithm still plays a key role in realizing the benefits of adaptive streaming.

The objective of our streaming algorithm is to take the SDUs produced by the Priority Mapper, and using their timestamp and priority labels, perform real-time adaptive streaming over a TCP-friendly transport. As it happens, our implementation of the algorithm works quite well over an unmodified TCP protocol.

The PPS algorithm works by subdividing the timeline of the video into disjoint intervals called *adaptation windows*. Adaptation windows are distinct from the mapper windows described in the previous section, an adaptation window consists of one or more mapper windows.

Figure 7 shows the conceptual outline of Priority-Progress Streaming. A pair of re-ordering buffers is employed around a *bottleneck*, which in our case is the TCP session. The buffers contain the SDUs of an adaptation window. The algorithm for Priority-Progress Streaming contains three subcomponents, the upstream buffer, downstream buffer, and progress regulator respectively. The upstream buffer admits all SDUs within the time boundaries of an adaptation window, these boundaries are chosen by the progress regulator. Each time the regulator advances the window forward, the

[8]Otherwise there can be pathological cases during streaming where low priority ADUs for one timestamp are kept even though higher priority ADUs with different timestamps, but belonging to the same mapping window, are dropped. For example, a P frame (low priority) might be kept when it's I frame (high priority) was dropped, however the P frame can not be decoded properly without the dropped I frame.

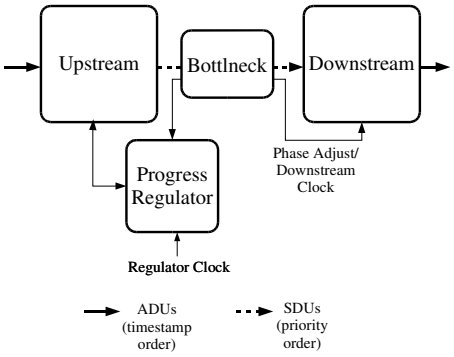


Figure 7: Priority-Progress Conceptual Architecture

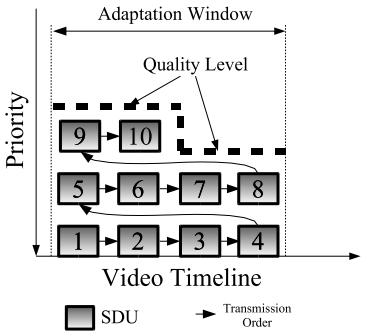


Figure 8: Adaptation Window Transmission: There are at most two quality levels per window.

unsent SDUs from the old window position are expired and the window is populated with SDUs of the new position. SDUs flow from the buffer in priority-order through the bottleneck to the downstream adaptation buffer, as fast as the bottleneck will allow. In order to sort into priority order, the buffer is implemented via a priority queue data structure. Similarly, the downstream adaptation buffer collects SDUs and re-orders them to timestamp order. When SDUs arrive late because of unexpected delays through the bottleneck, the progress regulator is notified so that it may avoid late SDUs in the future. The downstream buffer receives as many SDUs as the bandwidth of the bottleneck will allow and the rest, which are of lowest priority, are dropped at the server. In this way, the dropping will adapt video quality to match the network conditions between the sender and the receiver.

Window Number	Prepare		Transmit		Display	
	Start	End	Start	End	Start	End
1	0	1	1	2	2	3
2	1	2	2	3	3	4
3	2	3	3	4	4	5
4	3	4	4	5	5	6
5	4	5	5	6	6	7

Table 1: Priority Progress Example

As described in the paragraph above, each adaptation window goes through three distinct processing phases. The first phase is *window preparation*, which includes retrieval from the source (file or live capture), prioritization, and re-ordering from timestamp to priority order. The second phase is *window transmission*, where the SDUs are transmitted in priority order. The third phase is *decoding and display*. Table 1 gives a simple example for a sequence of five adaptation windows, where each row describes the timing of the phases for the nth adaptation window.

4.1 Responsiveness and Consistency

The basic premise of streaming is to start display as soon as possible relative to the start of transmission. Quality adaptive streaming has the added objective to adjust video quality so as to make full use of the available bandwidth, both to increase average quality and to prevent long term rate fluctuations from disrupting the stream entirely. However, it is also true that it is preferable to avoid exposing the user to visible quality changes. The size of adaptation windows in PPS determine important trade-offs between streaming latency, buffer space requirements, robustness to rate changes, and the consistency of quality. Smaller windows have the advantage of shorter startup delay, because the algorithm does not allow display of a window until transmission is fully completed. Larger windows have the advantage that quality will change less often, and larger rate fluctuations can be smoothed out.

In Priority-Progress, the sizes of the adaptation windows have a direct effect on the number of quality changes. Figure 8 shows how the final quality level, for a given adaptation window, is determined by the transmission order used in Priority Progress. The SDUs for the window are transmitted primarily in priority-order, and secondarily in timestamp order, as in the figure. So the transmission pattern is like filling the rectangle from left to right, bottom to top. In the end, there are (upto) two priority levels that have been reached, hence two quality levels, as shown by the dashed line[9]. Then in the limit, the total number of changes for the whole video is two times the number of adaptation windows in the video timeline. In this way, longer adaptation windows directly ensure more consistent quality, in that longer windows decrease the number of possible quality changes.

4.2 Window Scaling

The fact that shorter and longer adaptation windows each have their benefits reflects what is likely an inherent trade-off between responsiveness and consistency in adaptive streaming. However, it is not necessary to restrict all window sizes to the same value. The Priority-Progress algorithm includes the option to adjust the window size during the streaming process, which we call *window scaling*. With window scaling, the window duration starts out minimal, so that startup latency is minimal, and then the window duration grows with each new window as the stream plays. As the window durations get larger, the quality changes become less frequent. Compared to a fixed window duration, we will see that window scaling yields dramatically better balance between responsiveness and consistency.

Window scaling is possible because Priority Progress can transmit the video at a faster (or slower) rate than it will

[9]This assumes that quality for a single priority level is uniform, which is true for our priority mapper algorithm.

Window Number	Prepare			Transmit			Display		
	Duration	Start	end	Duration	Start	End	Duration	Start	End
1	1	0	1	0.5	1	1.5	1	1.5	2.5
2	2	1	3	1	1.5	2.5	2	2.5	4.5
3	4	3	7	2	2.5	4.5	4	4.5	8.5
4	8	7	15	4	4.5	8.5	8	8.5	16.5
5	16	15	31	8	8.5	16.5	16	16.5	32.5

Table 2: Window Scaling Example: windows grow at 100% rate

be consumed at the receiver. The consumption rate at the receiver is naturally fixed to the videos "real time" rate, but transmission schedule is not so constrained. The priority dropping mechanism is what affords flexibility in this respect. Sending a window faster just means that more SDUs may be dropped. In altering the transmission schedule, the Priority Progress algorithm can create (or reclaim) *workahead* in the streaming schedule, which is what allows subsequent adaptation windows to be larger (or smaller). Workahead accumulates whenever the duration of the transmission phase is shorter than the display phase. By definition, the transmission of the first adaptation window is a preroll window which establishes the initial workahead. With the exception of the preroll window, the accumulated workahead is the upper bound on duration of each step of the transmission phase. We call the ratio between duration of a transmission phase step and the duration of the corresponding display phase step the window scaling *growth ratio*.

Table 2 describes a timeline for five adaptation windows, where the growth ratio is fixed at 2. As in table 1, each row in the table describes processing for a single adaptation window. The columns show when the timing of the three phases for each window. We use a growth ratio of 2 here as it results in relatively simple numbers, but in practice we use more modest ratios. The results in the next section are based on a ratio of 1.1.

4.3 Priority-Progress Streaming Results

In this section, we describe experiments and results for PPS using our Quasar pipeline implementation. Our experimental setup consists of a group of Linux based PCs acting both as end hosts and as a router in a dedicated network testbed that implements a saturated network path. The router runs the NISTNet wide area network emulation package [24], which allows us to introduce artificial delay and bandwidth limitations. For the experiments presented here, we set the delay to produce a 50ms round-trip-time. We also set a bandwidth limitation of approximately 25Mbps and impose a queue length limit that matches the bandwidth delay product. For the entire duration of the experiments, the network is *saturated* with competing traffic.

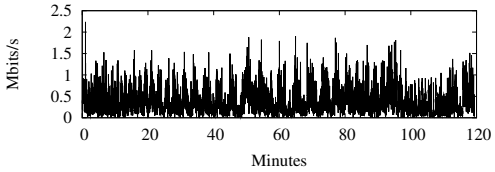
We have written a synthetic traffic generator, called `mxtraf` [20], that we use to generate the various levels and mixes of competing traffic. The mix is made up of non-responsive UDP traffic (10%), short-lived (20Kb) TCP flows (˜60%), and long-lived infinite-source TCP flows (˜30%), similar to measurements reported in [12] . Our experiments consist of streaming a two hour video through this saturated network path. To provide baseline performance references, we simulate two existing streaming algorithms assuming they are given the same video and available bandwidth from our experiments. The first algorithm is based on the Berkeley CMT, and the second on Feng's technique[3]. We then show the performance of PPS in two cases, the first using a fixed adaptation window, and the second with the PPS adaptation window scaling feature enabled.

Figure 9(a) shows the transmission rate of the TCP session used to transport the video. Figure 9(b) shows the maximum rate requirement of the video, which is significantly above the rate achieved by our TCP stream in the given conditions. For each streaming algorithm we show the frame-rate and SNR level achieved over the course of the whole stream[10]. Figures 9(c) and 9(d) show that the CMT algorithm has great difficulty with the conditions of our experiment. Video quality is extremely volatile, and there are several instances where the algorithm is not able to deliver even the minimum quality. Figures 9(e) and 9(f) show the sliding window algorithm fares much better, with fewer quality changes and no failures. Figures 9(g) and 9(h) show PPS with a fixed adaptation window behaves quite similarly to the sliding window approach. It would be possible to improve the consistency of PPS in the fixed window case by increasing the size of the window, but that would come at the direct expense of startup latency. The major benefits of PPS arise the adaptive window scaling is enabled, shown in figures 9(g) and 9(h), where quality gets more consistent over the course of the stream. In the majority of the movie, quality changes are several minutes apart, even though startup latency is in the range of 1 second.

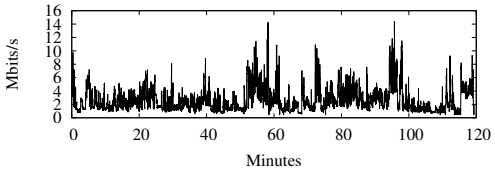
5. CONCLUSIONS AND FUTURE WORK

Streaming video over the Internet remains a compelling and challenging problem. While video compression addresses the issue of limited bandwidth, it is only recently that scalable compression has addressed the extra problem of highly variable bandwidth as on the Internet. Also recently, there has been consensus that video traffic should employ TCP friendly congestion control if it is to avoid threatening existing traffic and stability. In this paper, we presented a framework for adaptive video streaming centered around the simple concept of priority drop. We showed how, through priority-drop, to combine scalable compression and adaptive streaming in to form a very effective, tailorable, adaptive streaming system, supporting an encode-once, stream anywhere model. For future work, we are considering several extensions of the Quasar pipeline, including incorporating Priority-Progress streaming to an Application Level Multicast Overlay, extending Priority-Progress to inter-stream adaptation, and incorporating video compression with better and more scalability options.

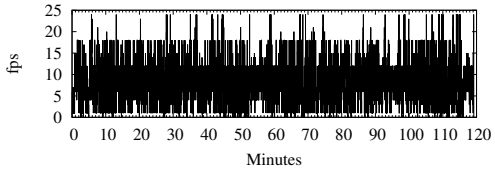
[10]Recall SPEG has four SNR levels



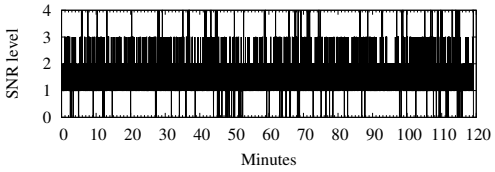
(a) Video stream TCP Transmission Rate (smoothed to 1s intervals)



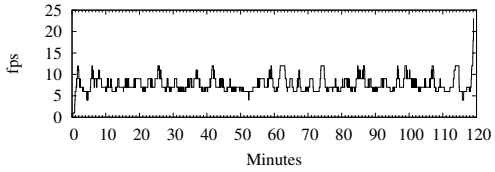
(b) Maximum Video Rate



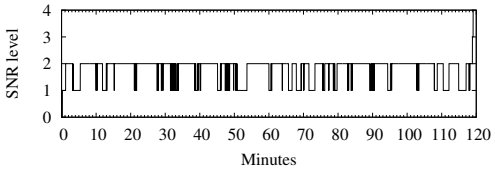
(c) CMT (2s buffer)



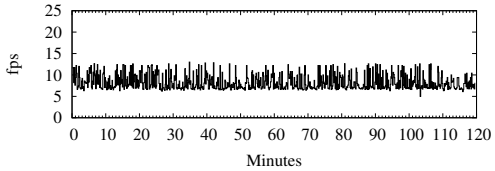
(d) CMT (2s buffer)



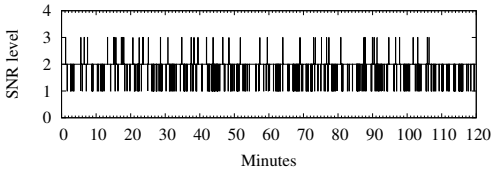
(e) Sliding Window Smoothing (60s window)



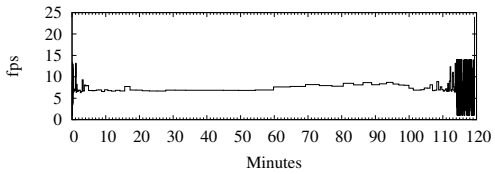
(f) Sliding Window Smoothing (60s window)



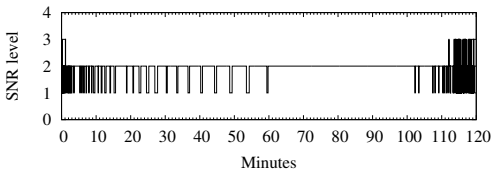
(g) PPS (10s window fixed)



(h) PPS (10s window fixed)



(i) PPS with adaptive window scaling (10%)



(j) PPS with adaptive window scaling (10%)

Figure 9: Sub-figure (a) shows the transmission rate in a saturated network over a two hour period. Sub-figure (b) shows the maximum rate of the video. Sub-figures (c)-(j) show the resulting video quality with each of four streaming algorithms.

6. REFERENCES

[1] B. Birney. Intelligent Streaming. http://msdn.microsoft.com/, October 2000.

[2] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole. A Distributed Real-Time MPEG Video Audio Player. In *Network and Operating System Support for Digital Audio and Video*, pages 142–153, 1995.

[3] W. chi Feng, M. Liu, B. Krishnaswami, and A. 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.

[4] G. Conklin, G. Greenbaum, K. Lillevold, and A. Lippman. Video Coding for Streaming Media Delivery on the Internet. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3), March 2001.

[5] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[6] N. Feamster, D. Bansal, and H. Balakrishnan. On the Interactions Between Layered Quality Adaptation and Congestion Control for Streaming Video. In *11th International Packet Video Workshop (PV2001)*, Kyongiu, Korea, April 2001.

[7] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, August 1999.

[8] S. M. W. Group. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. Technical report, World Wide Web Consortium, 1998. http://www.w3.org/TR/REC-smil.

[9] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. RFC 2543, March 1999.

[10] B. G. Haskell, A. Puri, and A. N. Netravali. *Digital Video: An Introduction to MPEG-2*, chapter 9. Chapman & Hall, 1997.

[11] Y. He, F. Wu, S. Li, Y. Zhong, and S. Yang. H.26l-based fine granularity scalable video coding. In *ISCAS*, 2002.

[12] G. Iannaccone, M. May, and C. Diot. Aggregate Traffic Performance with Active Queue Management and Drop from Tail. *Computer Communication Review*, 31(3), July 2001.

[13] 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, 1999.

[14] ISO/IEC. 13818-2 Information technology — Generic coding of moving pictures and associated audio information: Video . International Standard, 1993.

[15] ISO/IEC. 11172-2 Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 2: Video. International Standard, 1994.

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

[17] S. Jacobs and A. Eleftheriadis. Streaming Video using Dynamic Rate Shaping and TCP Flow Control. *Visual Communication and Image Representation Journal*, January 1998. (invited paper).

[18] S. H. Kang and A. Zakhor. Packet Scheduling Algorithm for Wireless Video Streaming. In *Packet Video 2002*, Pittsburgh, April 2002.

[19] J.-W. Kim, Y.-G. Kim, T.-Y. K. H.-J. Song, Y.-J. Chung, and C.-C. J. Kuo. TCP-friendly Internet Video Streaming employing Variable Frame-rate Encoding and Interpolation. *IEEE Transaction on CSVT*, 10, October 2000.

[20] C. Krasic, A. Goel, and K. Li. The MxTraf Network Traffic Generator. `http://mxtraf.sf.net/`.

[21] C. Krasic and J. Walpole. QoS scalability for streamed media delivery. CSE Technical Report CSE-99-011, Oregon Graduate Institute, September 1999.

[22] W. Li, F. Ling, and X. 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.

[23] S. McCanne, M. Vetterli, and V. Jacobson. Low-Complexity Video Coding for Receiver-driven Layered Multicast. *IEEE Journal on Selected Areas in Communications*, 16(6):983–1001, August 1997.

[24] NIST. The NIST Network Emulation Tool. `http://www.antd.nist.gov/itg/nistnet`.

[25] R. Rejaie, M. Handley, and D. Estrin. Quality Adaptation for Congestion Controlled Video Playback over the Internet. In *Proceedings of ACM SIGCOMM '99 Conference*, Cambridge, MA, October 1999.

[26] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, January 1996.

[27] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, April 1998.

[28] D. Sisalem and H. Schulzrinne. The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme. In *Proceedings of NOSSDAV*, Cambridge, UK., 1998.

[29] Unknown. Fast-start vs Streaming. http://www.apple.com/quicktime/.

[30] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, 1997.

[31] D. Wu. Streaming Video over the Internet: Approaches and Directions, 2001.

[32] N. Yeadon. *Quality of Service Filters for Multimedia Communications*. PhD thesis, Lancaster University, Lancaster, May 1996.