Multimed Tools Appl (2006) 30: 1–26
DOI 10.1007/s11042-006-0004-8

Adaptive filtering of MPEG system streams in IP networks

Michael Hemy • Peter Steenkiste • Thomas Gross

Published online: 24 June 2006

Abstract Congestion and large differences in available link bandwidth create challenges for the design of applications that want to deliver high quality video over the Internet. We present an efficient adaptive filter for MPEG System streams that can be placed in the network (e.g., as an active service). This filter adjusts the bandwidth demands of an MPEG System stream to the available bandwidth without transcoding while maintaining synchronization between the streams embedded in the MPEG System. The filter is network-friendly: it is fair with respect to other (TCP) competing streams and it avoids generating bursty traffic. This paper presents the system architecture and an evaluation of our implementation in three different operating environments: a networking testbed in a laboratory environment, a home-user scenario (DSL line with 640 Kbit/s), and a wide area network covering the Atlantic (server in Europe, client in the US). Moreover we examine the network-friendliness of the adaptation protocol and the relationship between the quality of the received continuous media and the protocol's aggressiveness. Our architecture is based on efficient MPEG System filtering to achieve high-quality video over best-effort networks.

Keywords MPEG System streams · Adaptive filtering · System architecture · Performance evaluation · TCP friendly

Effort sponsored in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

M. Hemy · P. Steenkiste · T. Gross (✉)
Department of Computer Science,
Carnegie Mellon, Pittsburgh, PA 15213, USA

Current address:
T. Gross
Department Informatik, ETH Zürich,
CH 8092 Zürich, Switzerland
e-mail: Thomas.gross@ethz.ch

1 Introduction

The infrastructure of the Internet provides an attractive platform for the distribution of all kinds of digital audio and video content. In addition to movies, we may want to transmit video clips retrieved from a database as well as teleconferences or lectures. However, distribution of continuous media poses a number of challenges. Since the movie is played as it is received, the transfer over the network must proceed at a specific rate to prevent buffer overflow or underflow at the player. If there is competing traffic in the network there is the risk of congestion, and consequently, packets may be dropped or delayed. Finally, if insufficient network bandwidth is available and we want to continue to smoothly play the movie, the application must reduce its bandwidth demands intelligently.

The key problem is to ensure that the player receives a continuous feed in the presence of variable congestion while maximizing the transfer of comprehensible information given the available (varying) bandwidth. Bandwidth reduction can be achieved by filtering the stream (selectively removing a part) before transmitting it on the congested link while maintaining the stream's key characteristics. There are numerous ways to filter a video stream: frame-dropping [4, 6, 10, 17, 26], low-pass filtering [26], color reduction [26], re-quantization [9, 26], and transcoding [2, 26] are some approaches; see Section 7 for a detailed discussion. Of all these approaches, dropping frames from a video stream, also known as *temporal decimation*, is relatively the easiest to implement and requires the least computational effort. In an MPEG video stream the problem that arises is not 'how to remove frames' but 'which frames to remove'. The inter-frame dependences inherent in the MPEG encoding standard have led to the development of prioritized transmission schemes. Kozen at al. [14] present algorithms for selecting the frames to drop such as to minimize the longest gap for a given bandwidth, where a gap represents the interval of unplayable frames.

However, when considering continuous media such as MPEG System streams (which include some number of video and audio streams), temporal decimation becomes a more complicated problem because the audio and video layers are multiplexed and encoded with synchronization meta-data. A straight-forward approach to demultiplex the System stream, filter the video, and remultiplex it is computationally expensive. Here we present a filter that can transparently remove frames from an MPEG System stream while maintaining audio and video synchronization. Since the output of the filter is also an MPEG System stream, there is no additional complexity required at the client-end to synchronize multiple separate streams.

A responsive adaptation process is the key to successful media presentation. Without it, the distortion caused by random packet losses in an MPEG stream will be propagated due to the inter-frame dependencies. The adaptation process can be controlled by various types of feedback. Some algorithms (the Vosaic player [10], the player of Li et al. [15], the VoD testbed [9], and OGI's player [6]), monitor packet arrival as well as information obtained after decoding the stream such as frame rate or delayed frames. In contrast to that approach we present an adaptation scheme based only on monitoring network packets. The latter has the advantage of reducing the client's complexity and completely decoupling the adaptation from the format of the continuous media. While we focus on the MPEG System format, the

basic concepts presented in this architecture, and in particular the adaptation process, can also be applied to other video standards.

The remainder of this paper is organized as follows. We first provide some background on the challenges of streaming MPEG-1 System streams over best effort networks. We describe the overall architecture of a system that adapts MPEG-1 System streams to the network conditions in Section 3 and discuss the design of a filter for MPEG-1 System streams in Section 4. We describe the implementation in Section 5 and present and analyze performance measurements in Section 6. Finally, related work is discussed in Section 7.

2 Background

A number of researchers have suggested the use of reservation of network resources to avoid congestion. Reserving bandwidth, however, is supported only by some networks but not yet provided by mainstream Internet protocols, and reservations often carry prohibitive costs or high overheads. The overhead of setting up a reservation may be tolerable if we play a full-length (i.e., 90 min) movie. However, we expect that distribution of many short video clips instead of a small number of large ones will become more common in the future. A multimedia database like the core of the Informedia system [12] developed at Carnegie Mellon contains a large number of short movie segments—video clips, sound bites from TV news, movie story boards—that provide access to individual scenes and shots, commercials, etc. To allow remote access to such a data collection, we investigate transmission of movies over existing best-effort networks that make up the Internet.

Several recent proposals for an "active services" architecture advocate the placement of computation agents within the network to address the congestion issue as well as to offer additional services. To be practical as an active service, a filter must be able to reduce the bandwidth of an MPEG System stream by removing frames from the video stream while maintaining the audio and synchronization information and while imposing only modest computation demands on its host. Additional requirements are that the filter outputs a valid MPEG-1 System stream so that we can use a standard player, that the video stream remains smooth, and that multiple levels of reduction are possible so we can better adapt to the available bandwidth. To allow wide deployment we require that only moderate computational resources are needed, i.e., filter must be efficient enough to run on a conventional PC. Even if there is no active service platform to host such a filter (i.e., the filter is placed on dedicated hosts), meeting these requirements is necessary for widespread use of such a filter.

This section provides background on the MPEG-1 System format and reviews why this format is very sensitive to packet losses in a best-effort network.

2.1 MPEG-1 System

MPEG-1 was primarily designed for storing video data and its associated audio data on digital storage media. As such, the MPEG standard deals with two aspects of encoding: compression and synchronization. MPEG specifies an algorithm for compressing video pictures (ISO-11172-2) and audio (ISO-11172-3) and then pro-

vides the facility to synchronize multiple audio and multiple video bitstreams (ISO-11172-1) in an *MPEG System*. MPEG-1 is intended for intermediate data rates in the order of 1.5 Mbit/s.

The video sequence layer of an MPEG video stream contains general information about the video stream (e.g., size, frame rate). The sequence layer contains groups of pictures (GOP), which consist of a header with information about the enclosed pictures (e.g., time stamp). The data contain the data portion of a GOP. An MPEG video stream distinguishes between I-pictures, P-pictures, and B-pictures—these pictures differ in the coding scheme. I-pictures (intra-coded pictures) or I-frames are self contained and are coded without reference to other pictures. P-pictures (predictive-coded pictures) are coded using motion prediction relative to the last reference picture and exploit immediate differences from temporally preceding I-pictures or P-pictures. B-pictures (bi-directionally predictive-coded) are based on a combination of previous and upcoming reference pictures. Since B-pictures are coded in reference to future pictures, the coding order must be altered such that decoders have enough information without overflowing when decoding the data and reordering it for display. While the MPEG format allows for an almost unbounded number of encoding schemes, most movies use one of two variants: they use either 24 or 30 frames per second (fps), with two periods (GOP) of 12 and 15 frames each. Inside each GOP, the first frame is an I-picture, every third frame is a P-picture; the rest are B-pictures.

An MPEG-1 audio stream consists of audio coded using one of three algorithms, which offer different levels of complexity and subjective quality. These algorithms are referred to as 'layers' in the coding standard. The coding algorithms use psycho-acoustic properties of the human hearing to compress the data (lossy compression).

The MPEG System layer is responsible for combining one or more compressed audio and video bitstreams into a single bitstream. It interleaves data from the video and audio streams, combined with meta-data that provides the timing control and synchronization.

While we present results for MPEG-1, which is an older standard, later versions of MPEG have a similar layered structure, so the same adaptation algorithms will apply.

2.2 Error susceptibility of MPEG-1 streams in best-effort networks

IPv4, today's Internet protocol, provides no support for resource reservation. Users are competing for bandwidth, and if a link becomes congested, packets are dropped. Since traffic conditions change continuously, congestion can start and disappear at any time. In the current Internet there is an assumption that it is the source's responsibility to reduce the data send rate when packet losses are observed to reduce congestion. For most applications, this reduction is done by TCP, the dominant Internet transport protocol, but if an application takes control of managing the send rate (as is the case for this application that uses UDP to transmit audio and video), it should also abide by this rule.

Random packet loss can hurt MPEG-1 System streams in two ways, besides the obvious fact that the information in the packet is lost. When we analyze random packet losses, we must take into account that network packets may not correspond to MPEG packets and that the latter are a layer completely separate from the video frames. The damage caused by the loss of a particular packet depends on its location

in the stream and on the robustness of the player in recovering from packet losses. In the worst case, we may loose a network packet that contains meta-data of the whole MPEG System stream (the MPEG System header), and players that rely solely on synchronization information found in the stream will be unable to continue. In a typical scenario, it is most likely that a packet lost will contain some part of a video frame with meta-data (video being the predominant stream).

In the context of the MPEG layers, a network loss translates into a disruption in the System pack layer and may result in concatenating parts from two different MPEG packets. This loss can induce corruption in the lower layers, e.g., corruption of the video or audio data. Part of the MPEG packet following the lost packet may be perceived as belonging to the previous MPEG packet, e.g., a network loss that affects an MPEG audio packet can corrupt the next MPEG video packet, thus damaging the video rendering even though the video data was not missing. If video data has been affected, the frame is decoded incorrectly. An incorrect I-picture or P-picture propagates problems to all dependent frames and corrupts these as well. In the worst case, we may loose a whole GOP, typically equivalent to half a second of video.

Figure 1 shows results of experiments with various levels of network packet losses for a collection of MPEG System streams. The darker area in the graph represents frames that cannot be decoded properly because of missing information. The lighter area in the graph represents frames that, while having all their data intact, cannot be decoded properly due to a dependence on damaged frames. Up to 35% of the total frames can be damaged by losing just 1% of the network packets. Similarly, Boyce and Gaglianello [5] observed that packet loss rates as low as 3% translate into frame error rates as high as 30%.

3 Architecture

In a typical video streaming application, there are two primary components: the client requesting the video and the server providing it. Typically, the server responds to requests from multiple clients. Our goal is to provide each client with the best

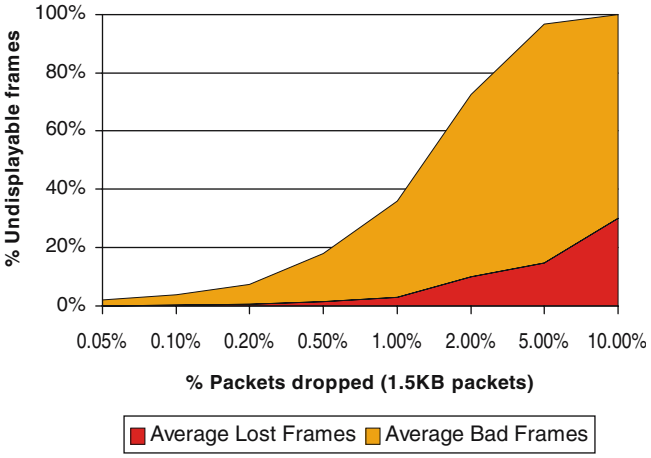


Fig. 1 Effect of network losses on MPEG System streams

possible video stream, subject to the network conditions on the path between the server and that client, while being fair to other users (connections) in the network.

The three components of our architecture are: a client, a server, and a filter. When there is congestion, the stream must be adapted by a filter. Such a filter could be placed with the server, but the video server may be too busy to handle the computation required to adapt the MPEG System stream to network conditions. If a single server provides multiple streams, it would be undesirable to restrict placement of the filter to the server node. In a multicast arrangement, each client may experience a different bandwidth, so in this case filters must be placed on nodes that connect to network bottlenecks to match a client's available bandwidth. Figure 2 depicts a multicast setup where a stream is transmitted to several clients. Filter placement algorithms [13] are beyond the scope of this paper, but here we assume that the bottleneck link of the server–client connection is between the filter and the client, and that the server-filter path experiences extremely low packet loss rates.

The filter-based architecture must meet a number of requirements:

1. adequately adapt to changes in available bandwidth;
2. maintain synchronization of the continuous media stream;
3. demand realistic computation resources, so that filters can operate on commodity platforms;
4. cooperate with other network usage;
5. behave transparently so that filters can be cascaded;
6. integrate into existing security models so that the filters can be deployed in the existing WWW infrastructure.

Requests for changes in the quality of the transmitted movie must originate at the client side. Such requests could either be entered interactively by the person watching the video, or preferably be triggered automatically by a process analyzing the received stream at the client. Unfortunately, perception-based models are computationally expensive. Since the client is already busy with decoding and rendering a continuous media stream, we need a simple, non-intrusive method to determine when, and how much, that stream should be filtered.

We chose to monitor the incoming network traffic and devised an algorithm that adjusts the degree of filtering to the current packet loss rate: the client only needs to send a binary signal to the filter (increase or reduce the stream). The filter picks the appropriate level of filtering based on the current level of filtering and the direction of the change request. This approach completely decouples the filter module from

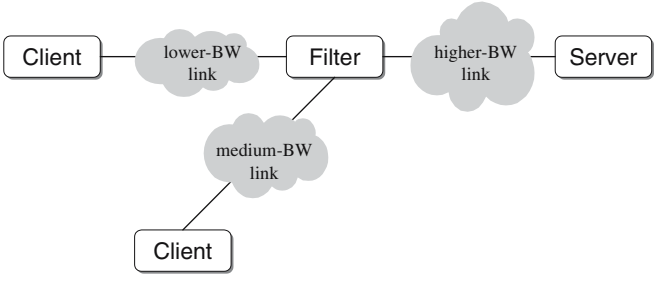


Fig. 2 Filter-based architecture

the network monitoring and feedback loop. The resulting software architecture is highly flexible with regard to what kind of filtering that is supported, and we may plug in filters for various continuous media formats.

The first three requirements are immediately satisfied by the MPEG System sensitive filter described in the next section. The fourth requirements is also met by our architecture but we do not discuss this aspect in detail. Cascadability is useful in multicast scenarios where multiple clients are viewing the same movie. Cascadability implies that there is a symmetry in format and control between the inputs and outputs of the filter. Additionally, the control protocol needs to propagate commands and replies between all the components on the chain. The client must be able to interact with both the server (e.g., on movie selection) and the filter(s) (e.g., on filtering requirements). The sixth requirement, to allow ubiquitous access, implies that we would like the client to be an applet that is running in a WWW browser. However, the security model for applets allows the applet to establish network connections only to the specific server from which it was downloaded.

We satisfy these last two requirements by maintaining two channels between every pair of components, one for control and the other for data, as indicated by figure 3. Control information is propagated along the chain in parallel with the data. When a component receives a command, it will execute the command (if it is destined for that component) or forward it (otherwise). Furthermore, the client applet is retrieved from the host executing the filter. With this design, the client must communicate with one host only (the filter), thus satisfying the Java security restrictions, and the control operations for clients and servers are independent from the number of filters.

4 Filter design

4.1 Frame removal algorithm

Removing frames from a video stream reduces the amount of data that needs to be transmitted to the client. While each displayed frame gets the same amount of playing time and is thus equally important, there is a big difference in both the size and information contents of each of the encoded frames. These differences are a consequence of the MPEG inter-frame encoding. I-frames are the largest, and B-frames are the smallest. While the relative sizes suggest that we should drop I-frames first, the information contents and inter-frame dependencies make this filtering impractical. The filter must maintain dependencies when removing frames while considering the smoothness of the resulting stream to prevent "stop and go" jerkiness. Kozen et al. [14] suggest algorithms for evaluating which frames to drop based on estimating the available bandwidth and minimization of the interframe gaps. Our

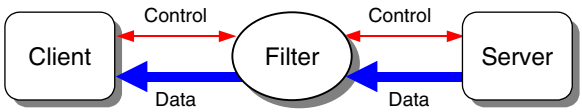


Fig. 3 Channels of communication in the network

approach does not require the filter to evaluate the available bandwidth. It just needs to be able, per request, to increase or decrease the streaming bandwidth.

To provide variable bandwidth, we define n filtering levels such that for $n = 0$ no filtering is performed, and for $n = M$ all the video frames are removed. In practice, the number of filtering levels is M' such that $M' < M$, because most players are not able to play an MPEG System stream that is defined to have both audio and video when its video stream has been completely removed.

To determine which frames can be dropped, the frame pattern must be known. The filter analyzes the first few GOPs in the video stream to determine the number of I-frames N_I, the number of P-frames N_P, and the number of contiguous B-frames N_B. For example, in the following GOP

$$IBBBPBBBPBBB$$

we have $N_I = 1$; $N_P = 2$; $N_B = 3$. After determining these constants, we can establish the various filtering levels.

The drop patterns for different levels are designed to conform to the dependences and to provide smoothness. Since B-frames do not have dependencies, the first level removes the middle B-frame from each group of contiguous B-frames. The second level removes two B-frames by selecting equally spaced frames. The third level (if $N_B \geq 3$) removes all the B-frames. E.g., in a GOP consisting of the following sequence:

$$I_1B_1B_2B_3P_1B_4B_5B_6P_2B_7B_8B_9$$

the first three filtering levels produce:

$$I_1B_1B_3P_1B_4B_6P_2B_7B_9$$
$$I_1B_2P_1B_5P_2B_8$$
$$I_1P_1P_2$$

The first levels of filtering do not reduce the data bit rate significantly since B-frames contain the least amount of physical data as indicated by figure 4. The actual frame rate that results from a given filtering level depends on the format of the movie, i.e., level 3 may result in 8 fps for one movie and 6 fps for another.

When all B-frames have been dropped, the bandwidth is further reduced by dropping P-frames. The next P-frame to be dropped is always the last one before the next I-frame. When all B- and P-frames are gone, we start dropping I-frames. At this point, the quality of the video degrades significantly. If we remove one of two I-frames (above all B- and P-frames), we remain with a video rate of one frame per second. Higher filtering levels can remove further I-frames, distancing them apart as much as needed to achieve the required bandwidth. The resulting stream looks like a slideshow at this point.

4.2 Efficient filtering of MPEG-1 System streams

Typically, if an underlying video or audio stream needs to be modified, the process requires demultiplexing the MPEG System stream, applying the filtering on the underlying data and remultiplexing according to the MPEG standard. MPEG multiplexing requires calculating various parameters (i.e., the *presentation time stamps* and the *decoding time stamps* of media units) to construct an MPEG System stream that allows a

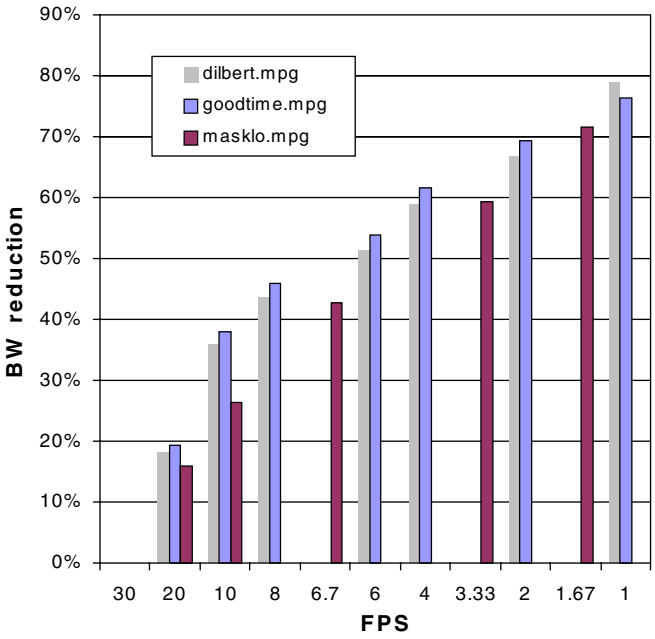


Fig. 4 Bandwidth reduction for various filtering levels expressed as different frame rates for stream with 24 fps

standard player to decode it on time. This process is obviously computationally expensive. Our design exploits the fact that a correct MPEG System stream already has the correct parameters, hence removal of data (namely frames) can be done while maintaining the corresponding pairs of meta-data and media units (video or audio).

The MPEG System stream is analyzed by a state machine that decodes rate information and identifies the System layers, the video layers, and the audio samples. Since the video sequence is broken into MPEG packets without considering frame boundaries, it is important to maintain state across all the MPEG layers simultaneously for all audio and video streams. When a frame of a particular video stream is detected, the filter checks whether it should be dropped or forwarded according to the current filtering level. The filter outputs MPEG packets containing only information that needs to be forwarded. With this method, there can be empty MPEG packets, and even empty GOPs, but keeping these empty (MPEG) packets provides the important benefit that synchronization is maintained, allowing in practice a client to decode and render the stream correctly. During this process the only change that needs to be made to the MPEG packet layer is the adjustment of the MPEG packet length.

To be practical, filtering should be efficient so a single server can support many clients. Figure 5 shows the cost of filtering a number of movies at three levels of filtering and compares those cost with the overhead of just demultiplexing the MPEG System streams. The costs are expressed as a percentage of CPU load of a Pentium 400 MHz machine while data is being streamed at video rates, and the costs only include the processing overhead but not the network communication. The results show that this particular type of filtering is very efficient and faster than any transcoding, since the latter would require demultiplexing the System stream as a first step.

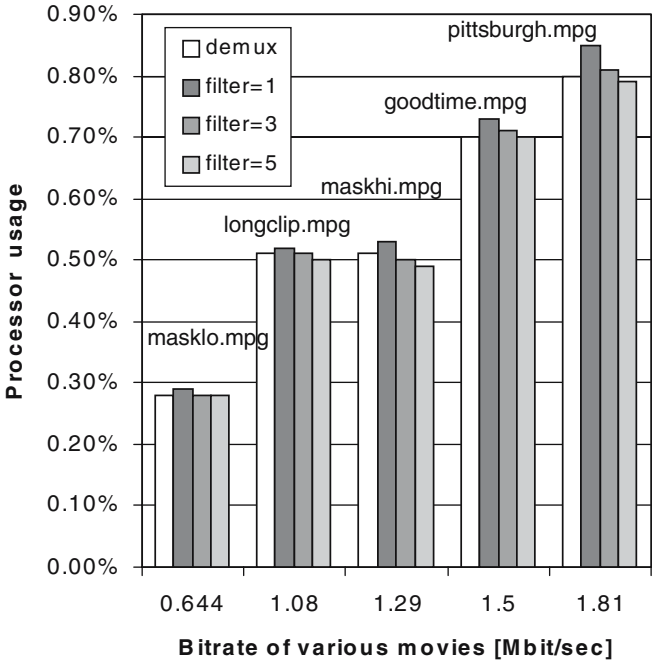


Fig. 5 Pentium 400 MHz load for filtering various movies

4.3 Adaptation algorithm

We have established in Section 3 that it is the client's responsibility to request modifications in the continuous media stream's bandwidth. To decide whether the filter should increase or decrease the bandwidth, the receiver continuously measures the current packet loss rate using a sliding window of variable length S packets.

During steady state operation, if the packet losses reach a threshold α during a window of length S_α, the client requests the filter to reduce the bandwidth. The value of α should be such that a packet loss of α (or less) in S_α still results in acceptable video quality. If after startup or an adaptation the threshold α is reached before S_α packets have arrived, the client immediately requests a bandwidth reduction.

A second threshold β, such that $\beta < \alpha$, and a window length S_β are used to determine when the bandwidth can be increased. If the packet losses are less than β during a period S such that:

$$S \geq 2^n * S_\beta$$

the client asks the filter to increase the bandwidth. Typically $n = 0$, however if an increase in bandwidth is followed by a decrease in bandwidth over a period S such that $S < S_\alpha$, then $n = n + 1$. Similarly if the opposite condition exists, then $n = n - 1$. For both cases we require: $1 \leq n \leq 5$. The reason for using two thresholds and a multiplier for S_β is to make bandwidth recovery less aggressive. Thereby the protocol is more friendly to competing traffic than an immediate increase. This behavior is somewhat similar to TCP's congestion control, that is, our protocol reduces bandwidth more aggressively than it increases it and adapts its

"aggressiveness" if attempts to reclaim bandwidth are not successful. We experimented with reducing the filter level by more than one in response to client requests to decrease bandwidth, but this did not improve the results.

After every request to increase or decrease the frame drop rate, the client temporarily suspends measuring the packet loss rate until it is notified by the filter that the drop rate change took place. This period depends on the number of packets 'en-route' and the frame being processed by the filter when it receives the request to change the drop rate.

4.4 Data streaming and smoothing

To allow the data channel to use UDP, it is important to limit the network packet size to avoid fragmentation. At the filter, the MPEG System packets are broken into smaller packets that fit within an MTU (the maximum transfer unit). We encapsulate the MPEG data in a datagram packet that contains a header with a sequence number, the current filtering level (required by the adaptation process) and some additional flags. By aligning the MPEG packet layer with the network packet, we are able to minimize the errors at the client when reassembling the packets.

The filter needs to send each packet at exactly the same time as the corresponding data packet from the non-filtered movie would have been sent. This rate is determined by the MPEG bitrate of the movie. When filtering frames, the remaining stream is not evenly spread across a standard GOP segment. In particular, when the filtering level is high, the stream contains only I-frames and audio. If those were to be transmitted at their corresponding time, the network traffic would be of a bursty nature. While the global average rate may be reduced by filtering, the bursty periods may suffer losses on a congested network, thus causing further reduction in the stream data. To provide better adaptation, it is necessary to smooth the network traffic. A smoothing algorithm tracks the amount of time that is 'saved' by partial frame streaming and statistically distributes it over the remaining data. Additional parameters provided by the control protocol allow fine tuning of the transmission rate to prevent buffer underflow or overflow.

5 Implementation

We have implemented the three-tier architecture with a filter as described in the previous section. All the components are implemented using Java for portability. In this section we discuss implementation issues relevant to each of the components.

5.1 Client

The client is based on the Java Media Framework (JMF), a package that supports the replay of audio and video streams in a browser. JMF supports a wide range of video and audio formats, including MPEG-1 System streams, and promises to be a widely used package to display multimedia material over the web.

JMF consists of two main components, a *player* and a *dataSource*. The *player* is responsible for replaying the audio and/or video stream and is typically optimized for a particular platform. The *dataSource* is responsible for retrieving the data. *DataSources* exist that retrieve data from disk, or retrieve data over the network

using a variety of protocols. We implemented a new *dataSource* to support our transport protocol described in Section 5.3. The filter for MPEG-1 System streams we describe has been used with JMF implementations from Intel and Sun.

5.2 Filter

As a provider of an active service (variable filtering), the filter must be able to handle multiple clients simultaneously. Also, there are no restrictions on the number of video-servers a filter could be uplinked to. Hence a filter behaves like a server and can actually provide an MPEG System stream from its own location without being connected to another video-server.

The filter responds to general setup requests as well as control requests to increase or decrease the bandwidth. Its responsibilities in the system can be summarized as follows:

1. receive MPEG System stream from server, or local host;
2. send paced MPEG System stream to client;
3. receive requests from client;
4. act upon requests or forward them to the video server.

5.3 Transport protocol

The control connection (figure 3) is used to exchange control information and is always based on TCP. MPEG data is transferred between the server and the filter as well as between the filter and the client using UDP. Since the filter needs the first few GOPs to be error-free to determine the filtering levels, and since the player implemented at the client's end needs to correctly identify the format of the continuous media to identify the filtering levels, we transmit the first couple GOPs (typically 24 or 30 frames) over the control channel, to guarantee reliable delivery.

5.4 Control protocol

The control channels are used for multiple purposes. Initially they are used by the client to request video clips and other information. To satisfy Java security requirements, the browser-based client can have only a single open connection, hence all requests are forwarded to the server by the filter over the control channels. The main use of the control channel is to carry the feedback, which is needed by the adaptation algorithm, from the client to the filter.

The control packets contain an opcode that identifies the request or information type, plus any relevant parameters. In addition, a destination tag encoded by an IP address identifies the target, allowing the propagation of commands or replies. Note that there are other ways of managing the control channel. Some of the control information could, e.g., be sent using RTP [21].

6 Evaluation

We present a detailed evaluation of our video streaming application. We first describe our experimental setup. We then compare the video quality with and

without filter-based adaptation. Finally we characterize the responsiveness of the filter with dynamic competing loads and the bandwidth sharing of our application with itself and TCP.

6.1 Experimental setup

We present results for three different scenarios. For all the experiments the client was run on a 200 MHz Pentium Pro machine.

- The first scenario consists of a dedicated testbed using 230 MHz Pentium Pro systems running FreeBSD as routers. The routers are connected using dedicated 10 Mbits/s Ethernet links that form the bottleneck. The filter is a 400 MHz Pentium II machine and the video server is a DEC 500/266. Several of the experiments use the H-FSC scheduler [23] that is part of the Darwin system [8] to control congestion at a finer grain. We set up a reservation of X Mbit/s (for example 2 Mbit/s) for continuous media streaming and one or more competing sources; the remaining link bandwidth is assigned to a UDP stream that floods the network but is prevented from using the X Mbit/s pipe by the scheduler. By changing X we can control the congestion conditions.
- The second scenario consists of a DSL connection between a central city location (campus of Carnegie Mellon) and a suburban home. The path between the server at Carnegie Mellon and client at the home has three segments: the campus local area network where the server and filter are placed, a T1 line to the telephone office, and a synchronous DSL line to the suburban residence. The last segment is the bottleneck link, its maximum UDP throughput was measured to be 649 Kbit/s. About 95% of this maximum bandwidth was available for the connection between filter and client.
- The last scenario involves a transatlantic connection between at client at Carnegie Mellon and a server at ETH in Zurich, Switzerland. We ran the server and filter on the same Linux box (a Pentium II 400 Mhz) for convenience. Since we use the Internet, we had no control over competing sources. Interestingly, the bandwidth for the transatlantic connection proved to be larger than needed for streaming video and all tries resulted in the whole movie being streamed without losses.

The adaptation parameters are set to

$$S_\alpha = 500; S_\beta = 500; \alpha = 25; \beta = 5$$

In Section 6.3 we report parameter sensitivity by experimenting with different values.

The video stream is received and displayed by a JMF-based client. While it is possible for us to subjectively judge and compare the quality of the received video stream, JMF provides no objective quality metrics. For this reason, we developed a second version of the JMF data source that is identical to the one we described in Section 5 but is an application instead of an applet. This setup allows us to store in files additional tracing information as well as the MPEG System stream for off-line analysis and replay. We also developed a tool (mpegOscope) that analyzes MPEG System streams and collects a variety of statistics, e.g., frequency of several types of

errors, lost frames due to the propagation of errors, etc. The mpegOscope tool is quite accurate, e.g., the number of good frames it reports is typically within 1% of the frames displayed. We verify this number by comparing statistics collected by the Windows Media Player when replaying the received stream.

When comparing different scenarios, we primarily use the following metrics: the first metric is the average rate of correctly received video frames. Two other metrics are the average bandwidth of the transmitted and received MPEG-1 System stream; these metrics capture the network resource utilization. The final metric captures the smoothness of the video stream. This is a fairly subjective metric, but the idea is that we want to distinguish between a video stream where every other frame is missing and a stream where the same number of frames has been dropped, but in a bursty fashion. The metric we use is the spectrum of frame rates over the length of the media stream; a video stream played mostly at one frame rate will be in general more smooth and pleasant than a video stream that has a broad spectrum of rates.

In the remainder of this section, we first present a detailed analysis of the effectiveness of our adaptive filtering technique for a single video clip. We then show how the different parameters affect the algorithm behavior. Finally, we look at the responsiveness of the algorithm with bursty competing traffic and at how the algorithm shares bandwidth with other video and TCP flows.

6.2 Evaluation of video quality

In this section we compare the quality of the received MPEG System stream with and without adaptation under some typical operating conditions.

Figure 6 shows the transmitted and received bandwidth for the video stream (in Kbit/s) for the DSL scenario using a movie with a bandwidth requirement of 1.07 Mbit/s. The first two cases compare the throughputs with the filter active and disabled, when the MPEG System is the only traffic in the network. We see that in both cases, roughly the same number of bits are received, but the transmit rate with adaptation is much lower than without adaptation. With adaptation performed by the filter, the output stays close to the maximum bandwidth of the critical link, marked by a horizontal line in figure 6. Without adaptation, the output rate is unconstrained and a large portion of the data is dropped along the way to the client.

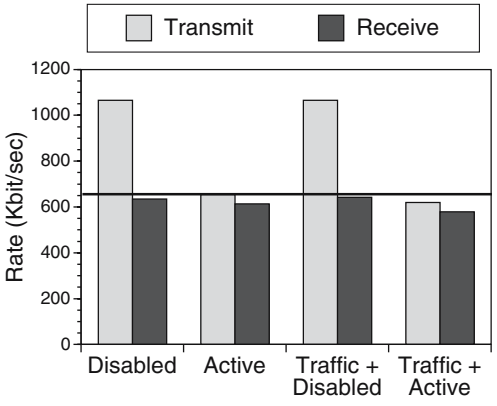


Fig. 6 Effect of adaptation on network load for DSL setup

Without filtering, the receive rate is equal to the link capacity. When filtering, the adaptation reduces the bandwidth until it reaches a steady state. We note that the average receive rate is slightly lower than link capacity. This is because frame filtering has only a certain distinct number of filtering levels as shown previously in figure 4. While the receive rates are almost similar with and without filtering, Table 1 shows that there is a big difference in the frame rates attained at the client, so the actual movie quality is significantly lower without adaptation.

To demonstrate that the frame rate achieved with adaptation (filter level 3) is optimal we perform two additional experiments: one with limiting the maximum filter level to 2 and one with statically configuring the filter level to 3 without any adaptation. The results are also presented in Table 1. A static level 3 filtering is optimal in that it provides the highest frame rate for the current network conditions. The adaptive filter comes close to this result when allowing adaptation to occur, only losing a few frames initially (until bandwidth reduction is achieved) and when probing for available bandwidth.

Figure 6 and Table 1 also show results of experiments we ran with competing traffic. For these experiments we use another machine in the home setting connected via the DSL line to browse the web and also issue file transfers (using FTP). The results represent an average of various runs. With the filter, the video stream is able to adapt, using less bandwidth than without the traffic. When the filter is disabled, the server continues to stream the continuous media at full rate, effectively flooding the DSL link and preventing any other traffic. While filtering allows the competing streams to proceed, most of the bandwidth is used by the video stream. The movie clip used for these experiments is too short to notice long-term adaptation in presence of competing traffic. We evaluate bandwidth sharing in Sections 6.4 and 6.5.

Figure 7 shows the frame rate distributions for cases one and two and compares them with the distribution for local replay. For each frame rate we show the percentage of all frames that are delivered at this rate to the client. We see that with adaptation the frame rates are centered around 5 and 10 fps, whereas without adaptation the peak is around 0 fps (i.e., no playable frames arrive at the client).

The MPEG movie used in these experiments has the following structure: *IBBPBBPBBPBB*. Removing all the B-frames (filtering level 2) still results in equally distributed frames $I_{BB}P_{BB}P_{BB}P_{BB}$, with a rate 1/3 of the original or 10 fps, where a subscript indicates that the frame was removed. The filter level needed for correct adaptation in this experiment is 3. At this level, the filter removes the last P-frame to maintain inter-frame dependences. The resulting stream $I_{BB}P_{BB}P_{BBPBB}$ has a frame rate of 7.5 fps, but the interval between the frames is no longer constant.

Table 1 Filter performance evaluation

DSL experiment	Average fps
Filtering disabled	0.5
Filtering active	6
Filtering limited to level 2	1.2
Filtering fixed at level 3	7.5
Traffic + Filtering disabled	0.5
Traffic + Filtering active	5

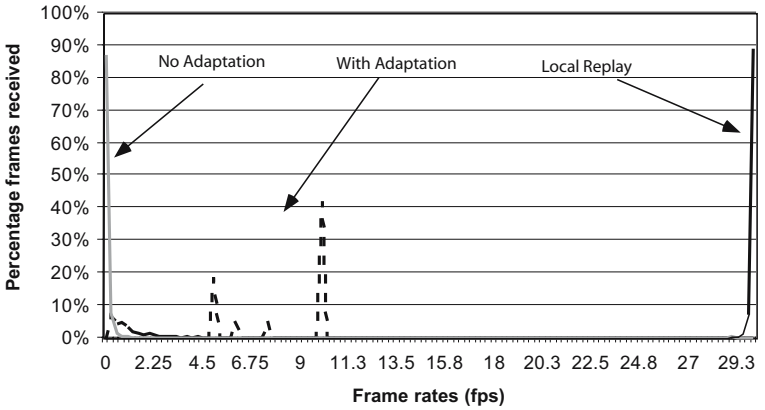


Fig. 7 Distribution of video frame rates for different experiments

The uneven spacing of frames causes the peaks in the frame rates distribution shown in figure 7.

Figure 8 summarizes the result of an evaluation obtained in the testbed. Figure 8a shows the transmit and receive bandwidth throughput for a 1.5 Mbit/s movie with a bottleneck link bandwidth of 1.1 Mbit/s. The adaptation behavior is similar to that on the DSL line: without the filter, a good part of the bits sent by the sender do not show up at the receiver. Figure 8b shows the distribution of good, lost/dropped, and damaged frames with and without adaptation. Clearly, adaptation yields a benefit also using this metric. Finally, figure 8c shows how many audio frames are correctly received with and without adaptation: again, adaptation pays off.

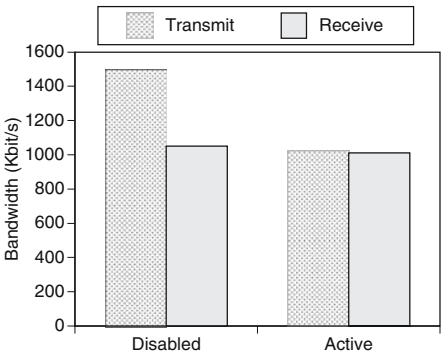
6.3 Parameter sensitivity

There are four parameters that can be used to control the operation of the filter: α, S_α, β, and S_β. We now report on the influence of these parameters. In addition, we provide data to justify the smoothing algorithm (Section 4.4).

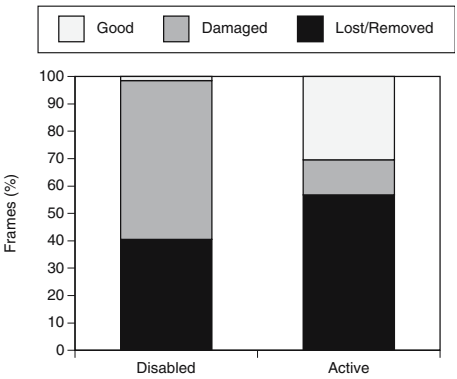
Experience has shown that α and S_β are the more sensitive parameters. In most cases α, the number of lost packets that triggers a request to decrease bandwidth, is reached before a period S_α, so usually $S_\beta = S_\alpha$. Since β is typically small, it has less of an impact than S_β on bandwidth recapturing since S_β enforces a delay equivalent to a window of at least that size before making a request to increase the stream.

It is clear that reducing α will cause a faster adaptation, but the question is "will a lower α result in more good frames?" Figure 9 shows the percentage of a movie's content (pittsburgh.mpg) that is *sent* by the filter (DSL setup described earlier, $\beta = 5$). If we inspect the results (with smoothing, filled symbols), we see that reducing the rate of adaptation (increasing α) increases the amount of data that is sent by the filter, since the filter can now tolerate longer periods of congestion before increasing the filtering level. However, the window size has a much larger influence on the behavior; a large window size implies that the filter will take longer to reclaim bandwidth. (With $S = 200, \alpha = 10$ the filter can in the best case tolerate almost twice as many losses as with $S = 1000, \alpha = 25$ before reducing bandwidth.) The filter is very effective in sending only those packets that have a good chance to

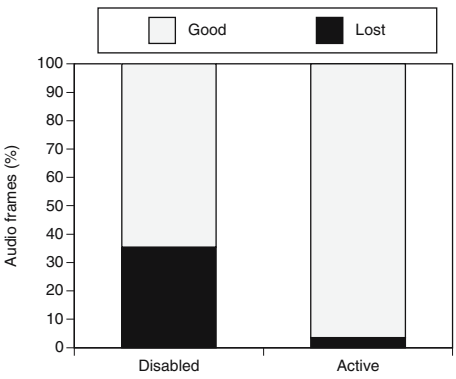
Fig. 8 Effect of packet losses in testbed



(a) Bandwidth.



(b) Video frames.



(c) Audio frames.

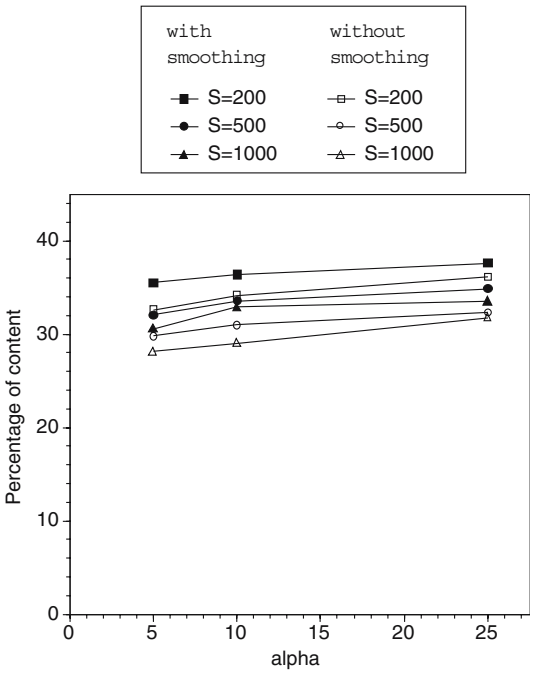


Fig. 9 Effect of filter (DSL setup, pittsburgh.mpg)

be delivered. The variation in the delivery rate (% *received*) for different values of α is masked by the measurement error, but increasing the window size from $S = 200$ to $S = 1000$ improves the delivery from 91 to 94%. Although a change in the percentage of content sent by 3% may not seem interesting, the net effect of increasing α yields about 4% more good data for $S = 200$ and 9\% more for $S = 1000$.

To further understand the contribution of α, we devise an experiment with constant competing load. We chose a competing flow with a bandwidth that would require an adaptation of two levels of filtering. In other words, removing all the B-frames from this movie should result in a video stream that would fit in a congested network where the load was constant. Table 2 shows the number of good frames received for various values of α for one movie (goodtime.mpg). The first row in the table is provided to allow a comparison with a loss-less scenario.

Table 2 Adaptation evaluation with constant competing traffic

α	Good frames	Max filter level
(No traffic)	5,988	0
40	2,580	2
25	2,591	2
15	2,605	3
10	2,555	3
5	2,524	3
2	2,476	4

Reducing α to 2 causes an average frame loss of an additional 2% of the full movie. To understand the reason behind this behavior we added the third column in the table (the maximum level of filtering reached). When α is reduced, the system becomes more sensitive to network losses. A few losses can cause the filter to reduce the stream further than is needed on average for the current conditions. The bandwidth will eventually be recaptured, but it will take more time to get to the correct level of filtering. Once the filter has activated a higher filtering level (removing more frames) than is needed, the requirement to wait a minimum period S_β results in lost transmission opportunities. As the filter attempts to be TCP friendly, S_β increases with increasing β, so these lost opportunities are the price of TCP friendliness.

For this experiment the optimal α is: $15 \leq \alpha \leq 25$. However a general characterization of α can only be done with additional experiments, possibly including additional parameters (e.g., number of hops between the client and filter). Determining right from the start the best value of α for a given movie and a given networking environment is still a topic of further research.

Figure 9 also depicts the influence of smoothing. Without smoothing a significantly smaller portion of the content can be delivered. Another view of smoothing (and the influence of the window size) is presented in figure 10. This figure depicts what fraction of all data sent is sent at a specific filtering level. Without smoothing, more data are sent at higher filtering levels, i.e., the filter adapts faster and stays longer at a more aggressive level of bandwidth reduction. The same effect is produced by increasing the window size. In our experiments, with smoothing the filter changes the level of filtering about twice as often as without smoothing.

As can be seen from figure 11, more than 50% of the data are sent at levels 4 or higher, so the filter encounters congestion that requires a bandwidth reduction. Figure 11 ($\alpha = 10$) shows the losses encountered when the filter operates at a given level. E.g., 37.0% of all packets sent in level 0 and 99.4% of those sent in level 6 reach the client ($S = 500$). It is not surprising that packets in levels 0 and 1 are lost frequently—while at this level, the filter is still trying to identify the appropriate stable state for this movie and networking scenario. Level 2 shows the benefits of a lossy memory: with a window size of 200, only packet losses in close formation trigger a change in filtering level. With a window size of 1,000, packet losses spaced further apart in time can force a change. However, the discrete nature of filtering

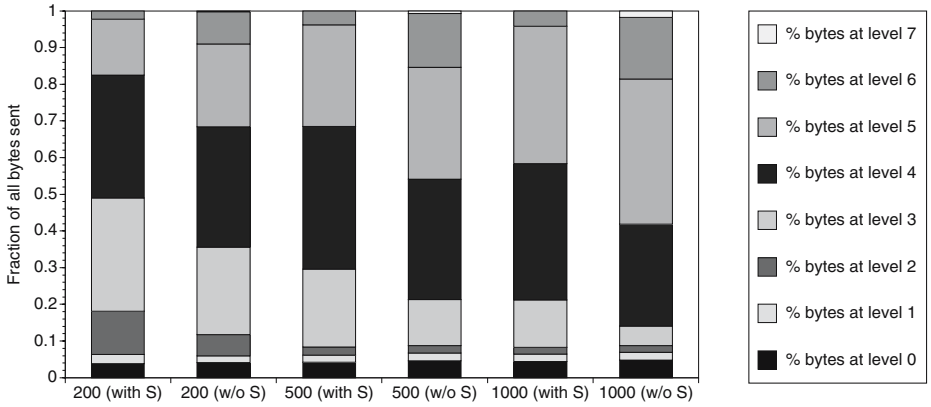


Fig. 10 Distribution of delivered packets (DSL setup, pittsburgh.mpg)

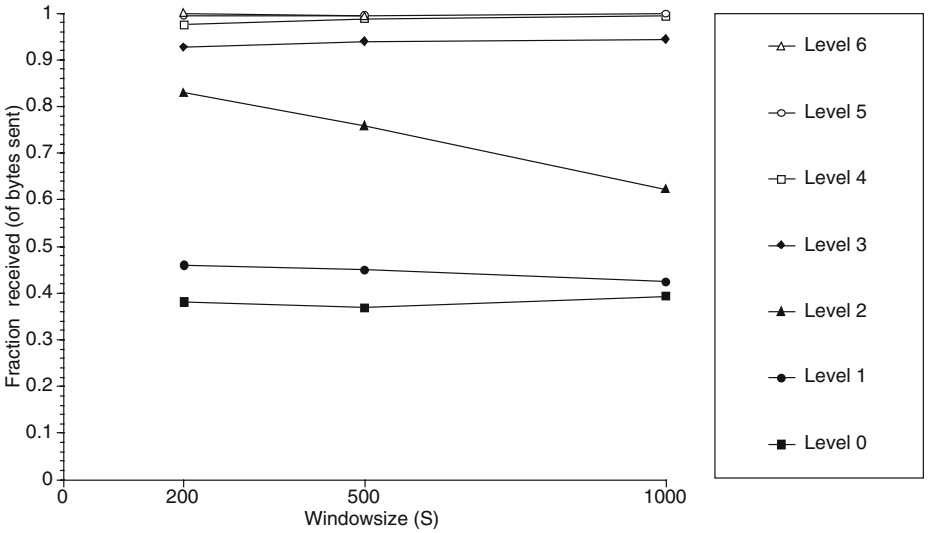


Fig. 11 Network effect on packets (DSL setup, pittsburgh.mpg)

works well. At level 3, the differences due to window size are small, between 92.7 and 94.3% of all packets arrive at the client.

6.4 Responsiveness

So far we have only considered competing loads that were constant or changing slowly, so the filter can track the network conditions and in general will be in a steady state condition. In this section we look at the behavior of the filter when the competing traffic is very bursty.

We set up a controlled experiment on our testbed where the video stream competes with a bursty stream on a bottleneck link with 3 Mbits/s of bandwidth. The video stream has a bandwidth requirement of 1.82 Mbits/s, contains a total of 76.98 MB of MPEG data without filtering, and has 10,140 frames (338 s 30 fps). The competing stream is an on-off stream consisting of bursts of 2.5 Mbits/s worth of UDP traffic separated by idle periods. The parameter settings for these experiments were: sliding window size is 500, low threshold is five, and high threshold is eight. The results of experiments for several burst and idle periods are summarized in Table 3.

The results for relatively short burst and idle periods (rows 1–3 in Table 3) are very similar. The filter aggressively drops frames (i.e., it uses a high filtering level), and as a result, very few packets are lost on the bottleneck link and the receiver receives few bad frames. This behavior is a direct result of the filter algorithm. At a beginning of a burst of competing traffic, the filter quickly increases the filtering level, thus matching the video bandwidth to the available bandwidth (about 0.5 MBs in this case). When the burst is over, the filter will slowly increase the frame rate to make use of the higher available bandwidth. However, it does this fairly slowly and the 5 s of idle time between bursts only allows it to reclaim some of the bandwidth. As a result, the video quality remains fairly stable around the quality supported by a

Table 3 Filter behavior with bursty competing traffic

Burst period (s)	Idle period (s)	Dropped frames (#)	Bad frames (#)	Sent MPEG data (MB)	Received MPEG data (MB)
2	5	8,078	214	27.29	26.76
3	5	8,354	178	23.35	22.99
5	5	8,535	167	21.92	21.09
5	7	2,257	1,356	69.37	63.81
5	9	2,453	1,287	68.41	62.64

little bit over 0.5 MBs of bandwidth. Note that the buffering inside the network helps in absorbing the impact of the bursts, which helps the video in achieving a throughput higher than 0.5 MBs.

When we increase the length of the idle period (rows 4–5 of Table 3), the behavior of the filter changes dramatically. With an idle period of 7 s or higher (for this set of parameters), the filter has enough time to reduce the filter level to 0. This results in many fewer dropped frames and a higher bandwidth consumption on the bottleneck link. We also see an increase in a number of bad frames, because at the start of every burst there will be mismatch between the video bandwidth (1.82 MBs) and the available bandwidth (0.5 MBs).

The filter behavior for bursty competing traffic is exactly what we want. For very bursty traffic with only short periods of high available bandwidth, we want the filter to settle on a frame rate that can be sustained. As the high bandwidth periods increase, we want the filter to start improving the video quality during these periods. We can argue that the results for rows 4–5 are not optimal (video quality changes too quickly). However, the competing flow is a worst-case scenario, and for purely reactive control, there will always be such a worst-case scenario. Moreover, we can always change the filter parameters to lengthen the high bandwidth period that is required to improve video quality.

6.5 Bandwidth sharing

In the next set of experiments we start to characterize how the adaptive video application shares bandwidth with other network users. We use our testbed and we set up a pipe with a bandwidth of 1.2 Mbit/s.

In the first experiment, we have an FTP stream compete with the streaming of a 1.5 Mbit/s movie for bandwidth on the 1.2 Mbit/s pipe. FTP is started first and can use the available bandwidth (1.2 Mbit/s). When the video streaming starts, the FTP stream slows down considerably as a result of packet loss and timeouts, as one would expect. After a while, FTP grabs back some of the bandwidth, and it peaks at about 600 Kbit/s. This competing FTP stream reduces the effective frame rate to 7.58 fps, compared to about 10 fps without a competing TCP (FTP) stream.

In the second experiment we have two video streams competing for the 1.2 Mbit/s of available bottleneck bandwidth. Under steady state conditions, the first stream gets about one third of the bandwidth and the second stream two thirds.

These results show that the video streaming filter leaves bandwidth to other users. While it appears that the video streaming is more aggressive than TCP, more experiments are needed to characterize this property more precisely.

7 Related work

Many researchers have looked at the problem of how to transmit multimedia data over best-effort networks. While all approaches use a *filter* that removes data as needed, they differ in a number of areas: (i) location of the filter, (ii) type of filtering applied, (iii) error recovery scheme, and (iv) adaptation algorithm. In this section we discuss related work along those four dimensions.

Location of filter. A filter can be placed either in the network or in the end-system. A number of researchers [2, 4, 26] present filters for video data that are located in the network. Since different applications require different filtering strategies, such network nodes need some knowledge about the type of data being filtered. Also, strategies that allow a client to find out about the location of filter nodes must be developed. RTP [21] proposes 'mixers' and 'translaters', which are placed in the network. The former ones mix streams and perform conversion between encoding formats; the latter ones translate across transport protocols (e.g., tunneling of a multicast stream into several unicast streams).

Berkeley's Continuous Media Player [17], OGI's distributed video player [6], and the Vosaic player [10] use the end-system to filter video data: frames can be dropped either at the sender (in case of a shortage of network resources) or at the receiver (in case timely display is impossible). An alternative is to have the adjustment of the frame rate done by the encoder [20], so video quality can be optimized using a priori knowledge of the available bandwidth. Finally, filtering can be done both at the server and in the network in a coordinated fashion [27].

Type of filtering. There are several ways to filter a video stream and to reduce its bandwidth: frame-dropping [4, 6, 10, 17, 26], low-pass filtering [26], color reduction [26], re-quantization [9, 26], and transcoding [2, 26]. Another approach is hierarchical filtering: the layering coding scheme for MPEG presented by Li et al. [15] multicasts three video streams. Each receiver subscribes to the base stream consisting of the I-frames. Depending on its capabilities, a receiver can additionally subscribe to the stream transmitting the P-frames or even to the stream containing the B-frames.

A number of researchers have also developed content-sensitive filters. Examples include adapting the filtering strategy to video contents [25], e.g., reducing frame resolution for high motion video while reducing frame rate for low motion video, and considering video contents when deciding what frames to drop [24], e.g., always keeping the first frame of a cut.

Error recovery. Lost data packets can be ignored, retransmitted, or recovered by a Forward Error Correction (FEC) scheme. The filter by Yeadon et al. [26] and OGI's player [6] ignore lost packets. The Continuous Media Player [17] pursues the second strategy by employing Cyclic UDP [22], which retransmits lost high priority data (i.e., I-frames in the case of MPEG video) to give them a better chance to reach the destination. In the Vosaic player [10] and in Columbia's VoD testbed [9], a client can demand retransmission of a lost frame. FEC is applied by Nonnenmacher et al. [18], where requests for retransmissions are handled by FEC transmissions.

Adaptation algorithms. Filter adaptation algorithms can be driven by video considerations (discussed here), while other work has more of a network focus (discussed below).

The Vosaic player [10] continually measures the rate of frames dropped by the receiver due to missing CPU power. If this rate exceeds 15% or falls below 5%, the server is instructed to lower, respectively, to increase, the frame rate. To cope with network congestion, the rate of frames dropped by the network is also measured and fed back to the server every 30 frames.

The player of Li et al. [15] also uses two thresholds to decide whether a client should subscribe to an additional multicast layer or whether it should drop one. The decision is made after receiving a GOP. In addition to the packet loss ratio, the number of late frames is also taken into account.

In Columbia's VoD testbed [9], the occupancy of the sender buffer is measured over 5 or 10 s intervals. In this way, momentary fluctuations due to the varying sizes of the different MPEG frame types can be overcome. The current occupancy is compared to the occupancy from the previous measurement. If necessary, the bitrate of the movie is adapted. Special care is taken to achieve convergence and to avoid oscillations around the desired rate.

In OGI's player [6], every component (server, network, client) can drop frames in case of missing resources. Additionally, the (filtered) display frame rate at the receiver is compared to the sending frame rate. If the difference is large, the sending frame rate is (linearly) decreased. In case of a small difference, the rate is (linearly) increased. The SCP (Streaming Control Protocol) [7] in OGI's player includes a TCP-like (window-based) congestion control to address the issue of TCP-friendliness.

Congestion control. A final area of related work is congestion control. The goal of congestion control is to ensure that traffic sources limit their transmission rate so they do not flood the network and bandwidth is shared by users in a reasonably fair fashion. Given that most of the traffic in the Internet uses the TCP transport protocol, the adaptation algorithm must guarantee that the video flows can coexist in a fair way with TCP flows. This idea of 'TCP-friendly' protocols, which was first mentioned by Mathis et al. [16], is at the heart of congestion for streaming media applications (e.g., work by Padhye et al. [19]). A flow is TCP-friendly if its arrival rate does not exceed the bandwidth of a conformant TCP connection in the same circumstances.

TCP uses an "additive increase, multiplicative increase" (AIMD) adaptation algorithm to deal with congestion [1, 11]. Specifically, it cuts its transmission rate in half when it experiences congestion and it periodically increases its transmission rate by a fixed increment to probe for more bandwidth. This adaptation algorithm does not work for streaming media applications since the multiplicative rate reductions will result in sudden drops in video quality that are very distracting to the viewer. For this reason, we use an "additive increase, additive decrease" (AIAD) algorithm that will result in more gradual changes in video quality. However, AIAD is much more aggressive than AIMD, so it would not coexist fairly with TCP. For this reason, we use the aging mechanism described in Section 4.3 to reduce the aggressiveness of the additive increase. Other groups have recently proposed similar techniques for TCP-friendly adaptation for streaming media applications, e.g. the SQRT algorithm in [3].

All of the projects described so far deal either only with video data or they transmit video and audio in two separate streams, thus requiring additional synchronization information for their playback at the receiver. Transmitting video and audio data in

one stream, as supported by the concept of MPEG System streams, has not yet been reported; this approach is at the core of the system that is described in this paper.

8 Concluding remarks

The MPEG format presents a number of challenges to a system that attempts to deliver MPEG System streams over a best effort network. However, given the amount of MPEG-1 movies available, supporting their delivery in their original format (without conversion or transcoding) is an attractive solution. The system presented here gives us an opportunity to evaluate an adaptive filter in scenarios with vastly different characteristics. The MPEG System sensitive filter provides an efficient solution that can be hosted on current mid-range PCs. Although we have an operational filter that performs as designed, a number of parameters deserve further study.

As new mid-range communication services (like DSL) become available to residential customers, an adaptive filter as presented here provides a cost-effective solution to allow homes to receive standard continuous media. The adaptation algorithm presented here is deliberately slower to recover bandwidth after congestion than it backs off in the presence of congestion, to ensure "TCP-friendly" behavior. Experiments with sharing have confirmed the importance of this design decision; cooperation with other network users is an essential property to scale up to Internet-wide deployment.

References

1. Allan M, Paxson V, Stevens W (1999, April) Rfc 2581: Tcp congestion control
2. Amir E, McCanne S, Katz R (1998, September) An active service framework and its application to real-time multimedia transcoding. In: Proceedings of ACM SIGCOMM '98. Vancouver, Canada, pp 178–189
3. Bansal D, Balakrishnan H (2001, April) Binomial congestion control algorithms. In: Infocom 2001. IEEE, Anchorage, Alaska, pp 631–640
4. Bhattacharjee S, Calvert KL, Zegura EW (1996) On active networking and congestion. Technical Report GIT-CC-96/02, Georgia Institute of Technology
5. Boyce JM, Gaglianello RD (1998, September) Packet loss effects on MPEG video sent over the public internet. In: Proceedings of ACM MULTIMEDIA '98. Bristol, England, pp 181–190
6. Cen S, Pu C, Staehli R, Cowan C, Walpole J (1995, April) A distributed real-time MPEG video audio player. In: Proceedings of NOSSDAV'95. Durham, New Hampshire, pp 18–21
7. Cen S, Pu C, Walpole J (1998) Flow and congestion control for internet media streaming applications. In: Proceedings multimedia computing and networking 1998 (MMCN98), pp 3310–3320
8. Chandra P, Fisher A, Kosak C, Ng TSE, Steenkiste P, Takahashi E, Zhang H (1998) Darwin: resource management for value-added customizable network services. In: Sixth International Conference on Network Protocols. IEEE, Austin
9. Chang S-F, Eleftheriadis D, Anastassiou D, Jacobs S, Kalva H, Zamora J (1997) Columbia's VOD and multimedia research testbed with heterogeneous network support. Journal on Multimedia Tools and Applications 5(2):171–184
10. Chen Z, Tan S-M, Campbell RH, Li Y (1995, December) Real time video and audio in the world wide web. In: Proceedings of Fourth International World Wide Web Conference. Boston, Massachusetts
11. Chiu D, Jain R (1989) Analysis of the increase and decrease algorithms for congestion avoidance. Comput Netw ISDN Syst 17(1):1–14

12. Christel M, Kanade T, Mauldin M, Reddy R, Sirbu M, Stevens S, Wactlar H (1995, April) Informedia digital video library. Comm. ACM 38(4):57–58
13. Karrer R, Gross T (2002, July) Location selection for active services. Cluster Comput (3):365–376. An earlier version appeared in Proc. 10th IEEE Symp. High-Performance Distr. Comp
14. Kozen D, Minsky Y, Smith B (1998, March) Efficient algorithms for optimal video transmission. In: Data compression conference
15. Li X, Paul S, Pancha P, Ammar M (1997, May) Layered video multicast with retransmission (LVMR): evaluation of error recovery schemes. In: Proceedings of NOSSDAV'97. St. Louis, Missouri
16. Mathis M, Semke J, Mahdavi J, Ott T (1997, July) The macroscopic behavior of the tcp congestion avoidance algorithm. Comput Commun Rev 27(3):67–82
17. Mayer-Patel K, Rowe LA (1997, February) Design and performance of the Berkeley continuous media toolkit. In: SPIE Proceedings Vol. 3020. San Jose, California, pp 194–2006
18. Nonnenmacher J, Biersack E, Towsley D (1997, September) Parity-based loss recovery for reliable multicast transmission. In: Proceedings of ACM SIGCOMM '97. Cannes, France, pp 298–300
19. Padhye J, Firoiu V, Towsley D, Kurose J (1998, September) Modeling tcp throughput: a simple model and its empirical validation. In: ACM SIGCOMM'98. ACM
20. Ramkishor K, Mammen J (2002, January) Bandwidth adaptation for MPEG-4 video streaming over the internet. In: 6th Digital Image Computing Techniques and Applications (DICTA). IEEE
21. Schulzrinne H, Casner SL, Frederick R, Jacobson V (1996, January) RFC 1889: RTP: a transport protocol for real-time applications. Request for Comments
22. Smith BC (1994) Implementation techniques for continuous media systems and applications. PhD thesis, University of California at Berkeley
23. Stoica I, Zhang H (1997) A hierarchical fair service curve algorithm for link-sharing, real-time and priority services. In: Proc. SIGCOMM'97, Cannes, SIGCOMM, ACM
24. Tan K, Ribier R, Liou S (2001, October) Content-sensitive video streaming over low bitrate and lossy wireless network. In: ACM Multimedia. ACM
25. Tripathi A, Claypool M (2002, March) Improving multimedia streaming with content-aware video scaling. In: Proceedings of the Second International Workshop on Intelligent Multimedia Computing and Networking (IMMCN). AIM
26. Yeadon N, Garcia F, Hutchison D, Shepherd D (1996, September) Filters: QoS support mechanisms for multipeer communications. IEEE J Sel Areas Commun 14(7):1245–1262
27. Zheng B, Atiquzzaman M (2001, May) TSFD: two stage frame dropping for scalable video transmission over data networks. In: IEEE Workshop on High Performance Switching and Routing. IEEE, pp 43–47

Michael Hemy is the President of CompuWiz Inc., which has been providing leading IT services in the greater Pittsburgh area since 1996. Prior to joining CompuWiz, Mr. Hemy was the President of Cineflo, a company providing streaming media solutions based on research performed at Carnegie Mellon University. Up until 2000, Mr. Hemy was a research scientist at Carnegie Mellon University where he supervised the development of networked applications in various fields such as medical imaging, chemical process optimization and video streaming. Mr. Hemy's work in the video streaming field was awarded a patent in 2004.

Peter Steenkiste is a Professor of Computer Science and of Electrical and Computer Engineering at Carnegie Mellon University. His research interests include networking, distributed systems, and pervasive computing. He received an MS and PhD in Electrical Engineering from Stanford University and an Engineering degree from the University of Gent, Belgium. You can learn more about his research from his home page http://www.cs.cmu.edu/~prs.

Thomas R. Gross is a Professor of Computer Science at ETH Zurich, Switzerland and an Adjunct Professor in the School of Computer Science at Carnegie Mellon University. He joined CMU in 1984 after receiving a Ph.D. in Electrical Engineering from Stanford University. In 2000, he became a Full Professor at ETH Zurich. He is interested in tools, techniques, and abstractions for software construction and has worked on many aspects of the design and implementation of programs. Thomas Gross has been involved in several projects that straddle the boundary between applications and compilers. And since many programs are eventually executed on real computers, he has also participated in the past in the development of several machines: the Stanford MIPS processor, the Warp systolic array, and the iWarp parallel systems. His current work in computer systems concentrates on networks.