Design of a Multimedia Player
with Advanced QoS Control

Rainer Koster

A thesis submitted to the faculty of the

Oregon Graduate Institute of Science and Technology

in partial fulfillment of the

requirements for the degree

Master of Science

in

Computer Science and Engineering

January 1997

The thesis "Design of a Multimedia Player with Advanced QoS Control" by Rainer Koster has been examined and approved by the following Examination Committee:

Jonathan Walpole
Associate Professor
Thesis Advisor

David Maier
Professor

Crispin Cowan
Research Assistant Professor

Acknowledgements

Table of Contents

List of Tables

List of Figures

Abstract

Multimedia applications are becoming more and more common. Possible features such as real-time access to remote data, combining information, and customizing presentations create a large potential for novel uses. On today's computers and networks, however, resources are still scarce for real-time processing of audio and video. Hence, good quality of service (QoS) management is necessary. The preferred quality as well as the appearance of a presentation may be user or task specific. Moreover, resources are shared among several users in unpredictable ways, requiring adaptation to varying resource availability.

This thesis describes the architecture of a real-time video and audio player integrating approaches to these requirements. It is based on a QoS model of independent content, view, and quality definitions [31, 33]. Authors can edit the content of presentations. Video and audio clips from different servers can be combined. View and quality is controlled in the two dimensions of temporal and spatial resolution. The user can customize the appearance by adjusting the view parameters of play speed and image size. Independently, frame rate and image resolution control the quality. A feedback mechanism provides automatic adaptation of the frame rate to resource availability.

This prototype demonstrates that advanced quality of service control and support for complex presentation can be provided and shows the implementation complexity involved.

Chapter 1

Introduction

1.1 Multimedia

The performance of today's computers and networks has made multimedia applications possible. However, they still easily exhaust the available resources. Multimedia presentations can include *static media,* such as text and images, as well as *continuous media,* such as audio and video. The latter consist of continuous data streams and have strict timing requirements and high throughput needs. Only sophisticated control and resource management allows meeting these requirements. Consequently, much multimedia research – including this thesis – focuses on continuous media.

These steady data streams need to be transmitted and processed in real time. For continuous media, late information is useless information. Particularly with video, this problem is aggravated by the data volume. The amount of information often approaches or exceeds the available bandwidth of networks or storage devices. The latter must provide large capacity, often at the price of slow speed. Compression techniques can be used to relax the throughput requirements. However, decoders need to be implemented in expensive special hardware or consume many CPU cycles, frequently moving the bottleneck to the CPU. Physically manipulating data before a presentation, for instance downloading an entire stream, also avoids some of the problems. However, this approach is not possible for large media items due to storage limitations. Moreover, it provides only slow access to data and may require processing large amounts of unneeded data, for instance, when retrieving an entire file while looking for particular information. Because of this combination of time constraints and bandwidth requirements, resources such as network band-

width, CPU capacity, or buffer space are still scarce, and must be managed carefully to avoid information loss and delays.

To control resource management it is necessary to measure and assess presentation quality. *Quality of Service* (QoS) here denotes performance characteristics of multimedia systems. Such characteristics include low level parameters such as throughput, transmission delay, and reliability as well as higher-level dimensions such as spatial resolution, frame rate (temporal resolution), and color depth. For continuous media streams, delay jitter and synchronization are important, too. Other elements of QoS include the cost of a presentation and priorities of different streams.

Like television, conventional applications play at most one video and one audio stream concurrently. Digital multimedia processing, however, allows composing *complex presentations* of several video and audio clips by concatenation and synchronization of streams. Complex content may become more and more common as users become familiar with the possibilities it offers. Advanced applications will need to support authoring and retrieval of such presentations.

In addition to resource scarcity, large multimedia systems have to deal with a distributed, heterogeneous, and shared environment. These systems are expected to use machines and networks (such as the Internet) that are shared with other users and conventional applications (such as compilers or word processors), resulting in contention for resources and, hence, variations in resource availability.

Current approaches address only some of these problems. PC-based applications can rely on a dedicated environment. They are designed for a particular type of hardware and operating system. Data is usually accessed only locally. Video-on-demand systems are distributed. However, they operate on dedicated special purpose hardware. State-of-the-art video conferencing tools can deal with a distributed, shared, and heterogeneous environment. They need to process live data with little latency and are specialized for that task. They do not support complex presentations or control over several QoS dimensions.

1.2   Motivating Examples

This section illustrates examples of advanced uses of multimedia with require-

ments that are not addressed by most current tools. Such applications need support for complex presentations composed of several video or audio clips and good control of several dimensions of quality such as spatial resolution, frame rate, or synchronization. Given that resource availability is unlikely to exceed the requirements of multimedia applications in the near future, QoS control is a useful technique for allowing efficient resource utilization. It achieves this goal by specifying appropriate priorities among different ways of allocating them. Moreover, these systems should be able to operate in an environment with shared resources. Although it is still hard to meet all these requirements with current hardware limitations, research needs to look forward and prepare for applications like these that are becoming more and more relevant.

1.2.1 Electronic News Gathering

Increasing storage and network capacities as well as the computational power of today's computer systems make digital TV production possible. Maier et al. [22] describe the architecture and functionality of a digital television studio: Media input and output devices, workstations for editing and control as well as multiple continuous media database servers are connected by a high-bandwidth network, which is linked to a public network. This architecture provides integrated support for functionality implemented by many dedicated data channels and specialized hardware in analog studios.

Incoming video and audio data is stored in the database. Because it remains immutably stored there, data can be shared easily by several users. Time stamps are included in the media streams. This labelling allows automatic resynchronization of two streams, typically video and the corresponding audio.

People composing news presentations need to select the clips to be included from some source files and combine them. Component clips may include new video and audio of the topical event, a commentary, an interview with an on-site reporter, archival data providing some background, and so on. It may be useful to retrieve several streams concurrently in order to select to most appropriate one; for instance, an event may have been captured with several cameras. Editing the news story involves, for instance, concatenat-

ing data streams, combining video and lip-synchronized audio, merging several video streams in one picture (e.g. for remote interviews), or adding voice-over narration to background material. A script describing the presentation in terms of operators such as synchronize, concatenate, or clip is created. The actual data should be included by reference while editing is underway.

During the editing process a variety of retrieval parameters needs to be supported. While browsing available source data for appropriate material, the play speed may vary from fast-forward to single-step. Low resolution can be sufficient for some stages of editing whereas others require higher quality. Since resources such as the internal network of the studio are shared by several users, it is necessary not to waste resources for high quality rendering when it is not needed. Hence, a way of specifying the needed quality of service (QoS) is required. Moreover, real-time constraints are harder to meet in a shared environment.

The link to the public wide-area network provides access to remote data. Material of other studios, archives, or correspondents all over the world can be used for news presentations. This type of network does not provide the bandwidth to transfer video data in production quality in real-time. The clips eventually required for the broadcast in high-quality have to be slowly downloaded before they can be displayed. While looking for appropriate material, however, a lot more data needs to be viewed. To do this efficiently, the news editor requires lower-quality, real-time access to remote material.

1.2.2 Sports

Staehli [31] describes another area of application for advanced multimedia systems: professional sports, using basketball as an example. Sports events are often recorded with several cameras and microphones providing multiple viewpoints and sound tracks[1]. Usually, these streams cannot be played back together in a synchronized way unless expensive TV production equipment is used. Moreover, to many users only the edited version with one camera and a mixed sound track is provided.

---

1. The local NBA team, the Portland Trail Blazers use three microphones and at least three cameras.

Storing this data digitally has several advantages. A variety of annotations such as player names or particular actions (goals, assists, and so on) can be added and used for interactive browsing. Professional sports teams, for instance, could use these features for evaluating the performance of their players and for studying other teams. Simultaneously displaying videos from several viewpoints can show details any one camera angle cannot.

Providing access to all video and audio recordings of an event may create a new way of watching sports in general. Users may have a small window for every camera and click on them to select what to see on the main screen. A commentator or statistics can be provided in additional windows. The continuing game can be displayed concurrently with a replay. To summarize, the viewer can have access to all information currently only available to the person editing the TV coverage.

When such complex presentations are transmitted over networks that do not have abundant bandwidth, specifying the required quality and, hence, resource consumption becomes necessary. In the example above, the small windows providing additional views need lower quality than the main window. For displaying action of the actual sports event, a high frame rate is needed and a lower resolution may be acceptable. A statistics board should be displayed with high spatial resolution to make it readable, but it needs to be updated only with a low frame rate. A picture of a commentator may not be important at all, but nice to have when resources suffice.

1.2.3 Security

Intelligence analysts need to access and communicate information efficiently. Advanced multimedia systems can facilitate this task. Networked computers can be used to transmit multimedia data from intelligence analyst to clients, from analyst to analyst, and from intelligence databases to analysts. If it is necessary to use shared, low-bandwidth networks to do so, the system has to provide means of specifying for what aspect(s) of presentation quality the scarce resources should be used. Maier and Walpole [21] give the assessment of the health of a foreign leader as a detailed example.

To detect changes in complexion and appearance of the leader, the analyst could

compare recent video clips and images with older ones. This comparison requires good spatial resolution and color depth. To demonstrate slurred speech in a TV interview the audio play-back needs to have high fidelity. Stiffness and slowness of movement in a recent video can only be discerned if the presentation itself has a high frame rate and low jitter, but a lower spatial resolution may be acceptable. To prove that a supposedly new recording is actually an old event from a different camera angle, both videos should be shown concurrently synchronized with each other. The analyst is likely to present this information to several clients on different sites providing different computer and network resources that may be shared with other users. Combining the material involved into a presentation including quality specifications makes sure that the important aspects of every part are most likely to be perceivable.

Another example is demonstrating that two recordings show the same person in order to prove that person's identity, for instance. It is necessary to play particular parts of either video concurrently to facilitate comparison. Also in this case, quality specifications can help to assign the right priorities: To compare the faces good image quality is required, to identify characteristic movements common to both recordings a high frame rate is more important.

1.3   Requirements

The examples above have shown a variety of requirements for this kind of advanced multimedia application:

*   They need to provide a good **QoS control**. For a video, important QoS dimensions are spatial resolution, frame rate (temporal resolution), and color accuracy. Delay jitter is critical for continuous media. If several streams are combined, synchronization is another category. Moreover, different parts of a presentation may be assigned different priorities. The user should be able to trade quality in some dimensions for quality in others.

*   Users should be able to combine data to form **complex presentations**. Operators such as concatenation, synchronization (concurrent presentation), and clipping (selecting particular parts) of streams need to be supported. Modifications of clips such as chang-

ing the audio gain, scaling a video image, or including a stream as slow motion are also useful.

- The content of a presentation can be **distributed**. Material from different remote servers may be included by reference in a presentation. During play-back, data must be retrieved from those locations in real-time.

- During a presentation, the user should be able to change **view** parameters such as play speed or image sizes.

- The application should work in large systems such as the Internet. One aspect of this problem is **heterogeneity**. There is a variety of operating systems and different types of network technology, for instance. Even if some of them provided good support for multimedia requirements, others may not. Hence, the application should not rely on features of particular computing environments. Moreover, **shared resources** need to be used, for instance network bandwidth and CPU cycles on a multiuser machine. Hence, resource availability may vary considerably. Even so, multimedia presentations should be possible, even when resources are scarce, and should provide a better quality, if resources are abundant.

- Network bandwidth and storage capacities will remain scarce resources for some time. At least for the large amounts of data needed for video **compression** must be provided.

1.4 Challenges

Today most networks, operating systems, and databases do not support resource reservation. Hence, it is impossible to determine or negotiate presentation QoS before playing. While retrieving and displaying data the quality needs to be adjusted to the currently available resources. Since many resources are shared, the activities of other users can lead to large and sudden variations in resource availability. Due to these variations, QoS must be adapted during a presentation as smoothly and quickly as possible.

In simple best-effort approaches every stage – of a video pipeline, for instance – simply drops data when there are insufficient resources to process it (not enough network bandwidth, buffer overflow, CPU too busy, and so on). This approach is very inefficient because all resources used for processing data that is dropped by a later pipeline stage are

wasted. A more sophisticated form of adaptation is needed.

Supporting complex content presentations and providing the user with control over view parameters requires a sophisticated system design. The application needs to run several video and audio pipelines at a time keeping them synchronized. Starting up and stopping as well as concatenating streams needs to be scheduled during playback. Connections to the servers of the respective clips need to be managed. In a single presentation, the quality of component streams can differ considerably, when some data is retrieved from a local server and some over a congested network.

Transmitting continuous media data in real-time over the Internet does not allow the use of a reliable protocol such as TCP. Because packets can be lost in the network, TCP tries to resend them. The resent packets, again, can be lost, and so on. Hence, transmitting a particular piece of data can take arbitrarily long. For continuous media, this delay is not acceptable. Not only is late data itself useless, but with a reliable protocol it also delays the delivery of subsequent data, potentially stalling the entire pipeline. Because of these problems, an unreliable, but fast protocol such as UDP must be used, and the application must deal with packet loss and out-of-order delivery.

Limited network bandwidth and disk space require compression at least of the video data. Most compression algorithms are lossy, making QoS management more complicated. Moreover, peculiarities of the technique used have a major impact on the design of the multimedia application. The MPEG standard for video compression, for instance, uses inter-frame decoding, such that some frames can only be decoded if surrounding reference frames are already available. Whenever a reference frame is dropped, all frames referring to it are useless, too. Any stage of the video pipeline should take these dependencies into account when making dropping decisions.

Because it is necessary for the user to control the trade-offs between several QoS dimensions, the adjustment of several parameters such as frame rate and image resolution must be supported. The goal is to find the configuration that provides the best quality possible using the resources that are currently available – with the notion of *best* being user and task specific. Coming close to this goal requires good heuristics. Moreover, even assessing quality based on a QoS specification is a non-trivial problem, which can be addressed using a formal error model.

1.5   Scope of the Thesis

Creating an advanced application with the requirements and problems described above is a major task that has involved several people in our research group. Richard Staehli [31, 33] defined a way of specifying QoS based on separate content, view, and quality descriptions. He has implemented a local player for uncompressed video, supporting complex content and user-controlled QoS adaptations. Shanwei Cen [4] has developed a distributed real-time MPEG video audio player that uses feedback to adapt to resource availability, but supports only simple presentations.

Based on their results, I have shown that support for advanced multimedia applications as described in Section 1.2 is feasible even with today's technology by developing an adaptive, distributed real-time player that supports complex content and user control over several QoS dimensions. The implementation integrated the additional features in Cen's player. The combination of requirements cannot simply be realized by a combination of individual solutions. That is, knowing how to build a distributed player and a complex content player does not imply knowing how to build a distributed complex content player. Many problems exist only for the combination: E.g., how to prefetch data for a clip starting up in the middle of a presentation? How to switch quickly between two remote servers? How to add per-stream controls to the user interface? It was necessary to re-engineer major parts of the system and design a new architecture.

While we are improving means of QoS control, it is not the subject of this research to perform the user-related studies to find out which QoS settings are optimal for particular users and tasks. The development of authoring tools or admission control are beyond the scope of this thesis, too. Moreover, it does not address video conferencing and multicast scenarios but focuses on the unicast transmission of stored data.

1.6   Outline of the Thesis

The next chapter gives an overview of related work, especially of the QoS model and the adaptation mechanisms constituting the framework on which the research for this thesis was based. Chapter 3 describes the architecture of the distributed video audio

player, providing the basis for the development of an application supporting the advanced features described above. In Chapter 4, the design and implementation of a new continuous media player are explained. Chapter 5 shows some of the player's performance characteristics. Chapter 6 discusses future work and Chapter 7 summarizes the conclusions of this research.

Chapter 2

Background

Since resources are still scarce for multimedia applications, it is generally not possible to provide a quality that is good enough for all users and all tasks at all times. Because of this limitation, exploring ways of assessing and managing performance of applications has been a central issue of multimedia research. The following sections give an overview of the notion of QoS on several levels and how they are related to each other. After that, the main aspects of resource management are discussed. Implementations of mechanisms such as admission testing, resource reservation, scheduling, policing, and controlled adaptivity have been proposed in order to improve the performance of multimedia applications. The degree to which performance guarantees can be provided depends on the system support for these resource management tasks.

Controlling several QoS parameters rather than having them hardcoded in multimedia players helps to exploit resources more efficiently by allowing finer adjustment to the needs of the user. To support compositions of several continuous media streams, presentation descriptions and synchronization mechanisms are needed. At the end of the chapter, the MPEG video compression technique and its effect on the design of a player are discussed.

2.1 Quality of Service

Due to resource scarcity, characterizing the performance of multimedia applications and hence the notion of QoS, is an important issue. Vogel et al. [39] and Hutchison et al. [17] give overviews of QoS in distributed multimedia systems. QoS has been discussed

in the literature on several levels. The following sections categorize them as *user-*, *perception-*, and *resource-level* QoS.

2.1.1 User-Level QoS

The ultimate goal of a quality notion is to represent the value of a presentation to the user. *User-level QoS* is an assessment based on several issues:

User Perception: Can errors in the presentation be perceived by humans? For example, it is likely that nobody can tell the difference between a video displayed with 50 frames per second and one with 60 frames per second. A single missing frame may not be realized either and need not imply a reduction of quality. Apteker et al. [1] and Steinmetz and Engler [35] have studied user assessment of quality.

User Preferences: Does the user care about certain errors? A user may see the difference between a 20 frames per second video and 10 frames per second, but may be perfectly happy with 10 frames per second, particularly when some cost is involved with a higher frame rate. Another user may feel bothered by a low frame rate, but may not care about low image resolution. Hence, perceived quality depends on subjective user preferences.

Task Dependency: What effect do errors have on the usability of a presentation for particular tasks? Reading figures and tables in an educational video requires a high spatial resolution but not a high frame rate, whereas it is the other way around for action videos such as sports. Other task-specific QoS requirements are discussed in Section 1.2 and in the literature [1, 39].

2.1.2 Presentation-Level QoS

Presentation-level QoS is independent of the user as well as of the underlying system and devices. It describes the output, not the required resources. A set of QoS dimensions can be defined on the presentation level [31, 33]. For videos, there is for instance frame rate, spatial resolution, and color depth. Delay jitter is critical for continuous media

streams. If a presentation consists of several streams, for instance a video and an audio stream, the accuracy of synchronization is another important characteristic. In contrast to the actual user perception, QoS can be described easily on this level. A technically literate user can use these parameters to specify his preferences and requirements. To facilitate this task, an interface also could provide examples for illustrating the choice [39].

2.1.3 Resource-Level QoS

Resource-level QoS parameters define quality in terms resources for particular devices, for instance CPU cycles; network throughput, bit error rate, and delay jitter; disk bandwidth; resolution and colors of the display. In communicating with devices (for measuring or requesting quality) and admission testers a multimedia system needs to use these dimensions [2, 3, 34, 41, 42]. They are mostly meaningless to end users.

2.1.4 Mapping between QoS Levels

Any sophisticated QoS management (with or without performance guarantees) needs a way to assess the quality a resource-level QoS configuration provides. This assessment is necessary to compare and choose among several possible configurations.

The mapping between resource and presentation-level parameters is often not a problem. For instance, for uncompressed video, the required bandwidth can be calculated by frame rate times pixel per image (resolution) times color depth. There may be, however, several ways of providing a particular presentation quality. The use of compression, for instance, trades CPU capacity for network bandwidth [31].

Assessing a presentation-level QoS specification, that is, mapping the presentation level QoS parameters to a quality value, needs to somehow take user perception, user preferences, and task requirements into account. Staehli proposes a general QoS model consisting of *content*, *view*, and *quality* specifications [31, 33]. Content describes the output related to logical space and time as edited by the author of a presentation. The user-defined view is the ideal mapping of content to output devices and real time. Quality denotes the

difference between actual and ideal presentation. An *error model* is used to calculate this difference. The error in each dimension can be interpreted as a combination of several error components such as shift, rate, jitter, and synchronization error. Weights for the components of all errors can be integrated in the interpretation, allowing calibration of the quality measurement for particular users and tasks. A multimedia application implementing such an error model does not depend on particular knowledge of user perception, because new information can be easily included in the error interpretation as it becomes known. This QoS model provides an overall quality measure that includes the relative importance of various QoS dimensions and allows the system to assess trade-offs between several configurations. A prototype architecture named SQUINT shows how this model can be used to control resource management on a local system.

2.2 Resource Management

Conventional resource allocation strategies mainly aim at providing fairness among tasks. Continuous media, however, introduce real-time constraints. In an overload situation, a fair policy will allocate an equal amount of resources to all tasks, but this share may be too little for every task, resulting in unacceptable quality for all of them. Because of these problems, new approaches to resource management have been proposed. They can be categorized by the level of performance guarantees they provide.

2.2.1 Performance Guarantees

Best Effort

Today's common operating systems such as UNIX and the Internet provide shared resources with little support for resource management techniques suitable for multimedia requirements. Hence, most applications have taken a best-effort approach in providing QoS. They adapt the quality of their presentations to the amount of resources that is currently available [4, 5, 18, 29]. Commonly, best-effort applications have a simple, one-dimensional quality model or no notion of quality at all.

Hard Guarantees

In a guaranteed performance scenario, the user specifies the desired QoS for a presentation. The system guarantees this quality by reserving resources such as CPU cycles, memory, network bandwidth, and storage system access and dedicates them to the application. If there are insufficient resources for meeting the specification, the system rejects the request. This approach requires the ability of network and operating system to reserve resources. A variety of protocols have been proposed for implementing this reservation. Some are listed in Section 2.2.3.

Soft Guarantees

Guarantees are not necessarily static. Allowing some degree of quality variation becomes useful if the resource needs of a presentation are not constant. This situation occurs when the QoS requirements change during a presentation or the effort for providing a certain level of QoS varies. The latter case is very common when compression algorithms are used. Depending on how well the content of a stream can be compressed the bandwidth and CPU cycles requirements change. In this case, hard guarantees require dedication of sufficient resources to meet the maximum need of an application. Often such a strong commitment is not needed and wasteful [15]. Other approaches use statistical guarantees or specify a range of acceptable quality parameters [6, 11]. This flexibility enables the system to reduce quality temporarily in transient overload situations resulting in more tasks being admitted and in a better resource utilization.

2.2.2   Resource Management Tasks

Several categories for resource management tasks and QoS processing steps have been identified in the literature [17, 39]. Many combinations of these steps may be useful. The effects range from statistical performance improvements to hard performance guarantees.

QoS Specification and Mapping

Approaches that monitor quality, or that predetermine resource consumption, need

a way of obtaining a QoS specification. The interface may simply consist of several sliders or it may let the user choose from several quality examples. The requirements may be input on any of the QoS levels described in Section 2.1. If user- or presentation-level are used, the specification needs to be mapped to the resource-level for communication with devices.

QoS Negotiation

If guarantees are to be provided, a feasible configuration must be negotiated between all system components involved. Each of them must check if it has the resources required to provide the requested quality. For instance, Figure 1 shows several resources
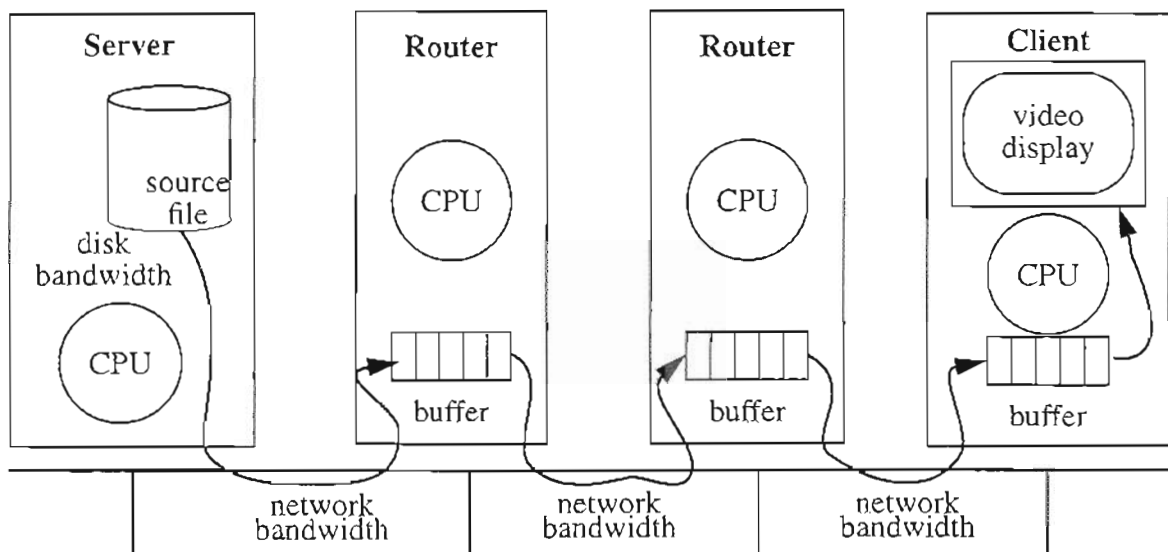
Figure 1: Resources in a video pipeline.

that may be needed for a distributed video player:

- Client CPU capacity
- Client buffer space
- Appropriate display
- Server CPU
- Appropriate source data
- Enough network bandwidth
- Sufficient processing capacity and buffer space at the routers.

Resource Reservation and Admission Testing

If all system layers have sufficient resources to provide the requested QoS, these resources can be reserved for the application. If there are insufficient resources, the new task could cause an overload situation. Hence, it will not be admitted, protecting the already admitted tasks from unacceptable performance deterioration. The user (or application) may choose a lower quality and try again.

Because performance guarantees require dedicated resources, admission testing is necessary to provide them. However, it is also possible to use weaker admission tests that allow overload situations to a limited extent or for a limited time, resulting in softer guarantees. Moreover, policing may be necessary to protect the admitted tasks from each other as explained below.

Scheduling

To provide guarantees, resources not only need to be reserved but also their use needs to be scheduled according to real-time constraints. Tasks handling continuous media streams need to be processed periodically by the CPU. Conventionally, operating systems do not support such requirements and implement some fair scheduling policy such as round robin. These scheduling policies can cause multimedia tasks to miss their deadlines causing low quality, even if there actually are sufficient CPU cycles to process all streams. This problem can be addressed by using rate-monotonic or earliest-deadline-first scheduling [34]. Split-level scheduling is an advanced technique using lightweight processes [12].

Policing

If it is possible for an application to consume more resources than requested at the admission test, it can try to obtain resources reserved for other tasks. Policing mechanisms can be used to detect these violations and react, for instance, by restricting the task's resource consumption or terminating it.

Monitoring

Most systems that allow QoS to vary implement some kind of monitoring of the currently provided quality. Based on this information, the user can be notified of quality

degradation or system components can be adjusted to a changed resource availability, for instance. Advanced adaptation techniques usually require monitoring.

2.2.3 Proposed Solutions

Protocols

A variety of network protocols for real-time transmission and bandwidth reservation have been developed. Guarantees can be provided only if all routers on a path through the network support a protocol, however.

The session reservation protocol **SRP** in the DASH system [2, 3] processes resources in two phases. In the first phase, all nodes in the pipeline reserve the resources involved. If a maximum end-to-end delay is exceeded, the reservation fails. On the other hand, if the delay is smaller than a target delay, the reservations are relaxed in a second phase.

The Internet Stream Protocol **ST2+** [8] was proposed as an adjunct to IP and can be accessed by higher-level end-to-end real-time protocols. It supports data streams to single or multiple destinations. Before the actual transmission, real-time channels are established. During this phase, a resource manager at each host or router reserves CPU, main memory, and network bandwidth according to QoS specifications.

The **Tenet** Group developed several protocols. The Continuous Media Transport Protocol (CMTP) [41] is based on the Real-Time Channel Administration Protocol (RCAP), a connection administration service, and the Real-Time Internetwork Protocol (RTIP), a network service providing real-time guarantees in ATM and FDDI networks. As for QoS, CMTP supports maximum limits of stream delay, delay jitter, granularity of a data loss, and probability of a data loss.

The resource ReSerVation Protocol (**RSVP**) [42] is designed to support not only point-to-point, but also multipoint-to-multipoint applications. Hence, it allows sharing of data streams. The application that reserves bandwidth is the only one to control the packets but may allow others to read the stream, too. This approach saves bandwidth in multicast scenarios. The receiver initiates the reservation.

For defining timing, multicast, and payload information, the Real-Time Transport Protocol (**RTP**) [30] is becoming more and more common.

Other Issues

In addition to the CPU and network, management of other resources such as storage access and physical memory also needs to have real-time capabilities [22, 32, 34]. Efficient IPC mechanisms, such as memory-mapped streams [12], are critical to process large amounts of multimedia data in time. Several integrated systems for providing end-to-end QoS guarantees such as the Meta-Scheduler [2], the Heidelberg Transport System (HeiTS) [9, 14], and the Lancaster Quality of Service Architecture (QOS-A) [17] have been developed. A QOS-Broker has been proposed as a way of encapsulating resource management [25].

2.3 Advanced Adaptive Mechanisms

The simplest method for adaptive resource management is a greedy best-effort approach. Presentation tools try to get as close to optimal quality as possible consuming as many resources as they can get to achieve this goal. This approach does not require any support by the operating system nor by the network and there is no need to deal with QoS specifications.

On one hand, this approach can result in an arbitrarily bad presentation. On the other hand, it aggressively consumes resources in order to provide an often unnecessarily good presentation [33]. In order to save resources for other applications or to be polite to others, the user may want to limit his resource consumption and be satisfied with a sub-optimal, but sufficient quality. A specification of a maximum quality can achieve this goal. Still, no support by the underlying system is needed to do so. However, the application must have a QoS specification and monitor the quality currently provided.

Simple best-effort applications adapt their resource consumption in an uncontrolled way. In a video pipeline, for instance, buffers overflow, network packets are lost, or a software decoder cannot handle the data volume, since the CPU is too busy. For information that is dropped at a late stage of the pipeline, all resources used for processing it in

earlier stages are wasted. An overloaded pipeline performs worse than a fully loaded one.

Feedback mechanisms can be used to adapt the rate at which the remote server sends frames to the rate at which they are eventually displayed [4, 5, 29]. Ideally, no more frames are fed into the video pipeline than can be processed with the available resources. Controlling this dynamic adaptation is not a simple task. If adaptation is too slow, variations in resource availability are not well compensated for. If adaptation is too fast, the system is susceptible to measuring errors and may overreact or oscillate.

Many applications use some kind of feedback. Often, however, their structure is specific to the application and their behavior is not well known. Cen et al. [4] have taken a more systematic approach by using a feedback toolkit for adaptation in their video player. The toolkit contains several filters and control algorithms [23, 27]. Using well known components allows the composition of predictable feedback mechanisms based on control theory.

2.4   Controlling Several QoS Dimensions

For video, the simplest parameter to vary in order to change resource consumption is the frame rate. Having control over several QoS dimensions provides a more exact way to specify quality (for guaranteed performance approaches) or to adapt quality (for best-effort approaches) [9, 36, 38]. Such dimensions include spatial resolution, color depth, or the lossiness of compression algorithms.

If a multimedia application is capable of varying several QoS parameters, it needs a way of assessing the trade-offs between adjusting in one QoS dimension or another. An error model [31, 33] as described in Section 2.1 can solve this problem by providing an overall quality measure based on user perception and preferences. A quality value can be calculated for QoS configurations, creating a partial order among them. The system can assess which of two possible configurations is *better*. Another approach has been proposed by Thimm and Klas [37]. They describe possible ways of adaptation as δ-sets and propose a heuristic scheme for selecting the most appropriate one, based on resource availability and user preferences.

2.5 Complex Presentations

Non-distributed complex multimedia presentations have been used for several years, for instance for producing educational software. For authoring and playing composed content several description and synchronization mechanisms have been proposed. Examples are Muse [16], Object Composition Petri Nets [20], MAEstro [10], the MHEG-model [24], structure-based authoring [13], or a video algebra [40].

With complex presentations synchronization between continuous media streams becomes an important issue. It is not only necessary to play several streams simultaneously (video and lip-synched audio for instance) but also the start-up time of a stream relative to others is critical. Particularly when retrieving data from more than one remote server, precise start timing is hard to achieve due to variations in network latency for transmitting the command. Hence, starting streams from different sources simultaneously or starting a stream at some given time during an ongoing presentation requires the application to plan ahead and, for instance, to prefetch the beginning of a stream. Such preparation for future events can be derived from a description of the presentation content.

If several concurrent continuous media streams are supported, resource management becomes more complex, too. The available resources not only need to be split among quality dimensions, but also among streams. The QoS specification should include the user's priorities. Compton and Tennenhouse have addressed a similar problem for several collaborative applications [7].

2.6 MPEG Video Compression

The video player presented in this thesis uses the MPEG encoding [19]. This lossy technique uses intraframe as well as interframe compression: Intraframe compression encodes an image by itself, interframe compression encodes images relative to others exploiting temporal locality of a video. I-frames are intraframe encoded only, P-frames depend on the preceding I or P-frame, and B-frames depend on the preceding as well as on the succeeding I or P frame. The pattern of these frame types is not the same for all streams, a typical example is IBBPBBPBBPBB. These dependencies have a considerable

effect on the architecture of an MPEG player. Transmission and display order of frames are different, because reference frames that are later in presentation order need to be transmitted and decoded earlier than the referring frame. Starting at a random position requires either starting at I-frames only or transmitting all reference frames required. More importantly, dropping frames at overloaded pipeline stages and temporal scaling, that is sending with a lower frame rate than the source file, needs to be done in a sophisticated way. If an I-frame is dropped, no frame up to the next I-frame can be decoded. Moreover, the MPEG encoding does not support scaling in other dimensions.

2.7   Summary

To control resource management, a QoS notion is necessary. QoS can be specified in terms of device parameters (resource level), characteristics of the output (presentation level), or an assessment modeling the value of a presentation to the user (user level). Throughout this thesis, most quality notions are expressed at the presentation level. An error model can be used to interpret a presentation level description and map it to user level QoS.

Providing good support for the resource management needs of multimedia applications requires extensions of operating systems and networks by new resource management features. With support of all stages of a continuous media pipeline performance guarantees are possible. Since this support is not common yet, however, an adaptive resource management approach needs to be taken. Feedback mechanisms allow doing this adaptation in a systematic and controlled way improving resource utilization. Applications that can control several QoS dimensions are capable of adapting their resource consumption in different ways approximating the needs of the user more exactly.

Supporting presentations that are composed of several streams from possibly different servers requires coordination of timing and resource management among the streams. Such complex presentations introduce the need of additional synchronization and prefetching mechanisms.

Chapter 3

Initial Architecture

3.1 Overview

A distributed adaptive video audio player has been developed by Cen. An earlier version has been presented at the 5th NOSSDAV workshop [4]. The program has been
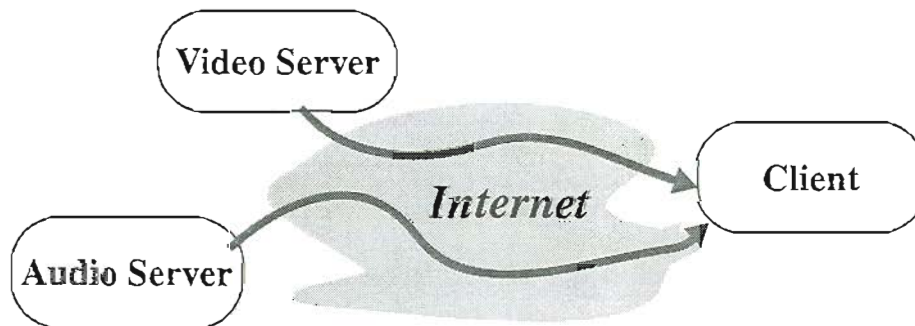
Figure 2: Player Architecture

implemented in C and ported to several UNIX platforms. The design of this player has been influenced by the Berkeley Continuous Media Player [28, 29] and uses the same software MPEG decoder as that player [26].

The overall architecture is shown in Figure 2. The client receives data from a video server (VS) and an audio server (AS) via the Internet. The servers may be located at different hosts. Of course, they also can be run locally on the same machine as the client. The data is read from files by the servers. The video data is MPEG-1 encoded and is displayed using the X Window System. Audio is uncompressed (μ-law format) and can be output to AudioFile or native audio devices.

3.2 Quality of Service

The player differentiates between play speed and frame rate. The user-specified play speed controls the mapping of video frame sequence numbers to real time using the client's system clock. The number of frames displayed per second can be varied independently, providing a means of quality control. For instance, when playing a video that is recorded at 30 frames per second at twice the regular play speed, it is not necessarily played at 60 frames per second. It is rather possible to keep the quality (that is frame rate) constant and omit every other frame in the source file. That implies, according to Staehli's QoS model, orthogonality of view (play speed) and quality (frame rate) is supported. However, time is the only variable QoS dimension. The synchronization between video and audio is achieved by playing a block of audio samples together with the corresponding video frame. As described below, feedback mechanisms are used to keep server and client clocks synchronized. The timing of both streams is controlled by the audio device.

By specifying a desired frame rate, the user can define a maximum quality and hence limit resource consumption. There are no quality guarantees however, and the frame rate actually achieved depends on the amount of resources available. Figure 3 illustrates the mapping of frame numbers to time according to play speed and desired frame rate for a video recorded with 30 frames per second[1]. If resources are insufficient not all of these frames can be displayed, however.

Real time (msec)		0		55			110			165		220			275			330
Play Speed	Frame Rate																	
100%	30*	1		2	3		4	5		6	7		8	9		10		11
100%	15	1			3			5			7			9				11
150%	45*	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		16
150%	30	1	2		4	5		7	8		10	11		13	14				16
50%	15*	1			2			3			4			5					6

*The result is the same for higher frame rates.

Figure 3: Frames over Time

---

1. This mapping assumes that there are no inter-frame dependencies. Such dependencies exist for MPEG and are discussed later.

3.3 Architecture

For every requested connection a main server process on a remote host forks a child process, a video server (VS) or an audio server (AS). These per-stream processes operate independently of each other and the main server process. For every stream, there is a TCP control connection and a UDP data connection. The client sends commands such as 'play', 'rewind', or 'stop' to the servers. When playing, VS and AS periodically read data from disk, send a packet over the UDP connection, and block for a certain length of time. The timing can be adjusted by a feedback mechanism as described in the following section. Since audio requires little bandwidth compared to video, and gaps in an audio stream are easily perceived by the user, the player tries to resend lost audio packets once. Tables 2 and 1 give an overview of the protocol between client and servers.

Table 1: Protocol Data Connection

Data Connection UDP	Video	Audio
data packets (to client)	position, frame type, MPEG reference frames, *video frame* [Data packets may be chopped into several network packets.]	position, number of samples, *audio samples*
feedback packets to server	clock offset (adjust server clock), clock rate change, send frame rate, MPEG header request	clock offset, clock rate change, data resend requests

To locate frames in the MPEG video file, VS uses a frame index that includes frame sizes, frame position in the file, and information about MPEG headers and groups of pictures. To generate this index, the MPEG stream is parsed the first time it is opened. Because this parsing can take several seconds, the index is saved on disk for future accesses. When a video connection is established, the index is loaded into main memory.

Figure 4 shows the architecture in more detail. The client consists of five processes: The video buffering (VB) and audio buffering (AB) process receive video and

Table 2: Protocol Control Connection

Control Connection TCP	Video		Audio	
Commands (to server)	Parameters (to server)	Reply (to client)	Parameters (to server)	Reply (to client)
init	filename	number of frames, image size, frame pattern	filename device information	number of samples, format information
close connection	–	–	–	–
play	position, speed, send pattern	–	position, speed	–
play speed	speed, send pattern	–	speed	–
fast forward, rewind	position, speed	–	(no audio)	
step, position	position	–	(no audio)	
stop playing	–	–	–	–

audio packets and put them into buffers. The buffering is necessary to remove network jitter. VB also reassembles video frames that have been chopped into several network packets. The video decoder process (VD) does the MPEG decoding and dithering into images ready for display. These images are put into another buffer (B2). The buffers are located in shared memory, allowing data to be passed to other processes quickly. Access is synchronized with semaphores. The user interface process UI maintains the connection to the X Server. It displays images and manages the user interface. The control process CTR handles the actual timing and releases the video frames from B2 to UI to be displayed. In addition, it transfers the corresponding audio samples to the audio device. CTR and UI communicate via sockets, passing frame display commands in one direction, and user button commands in the other. Global status variables are stored in shared memory and can be accessed by all processes. Signals are also used as a mechanism for inter-process communication.
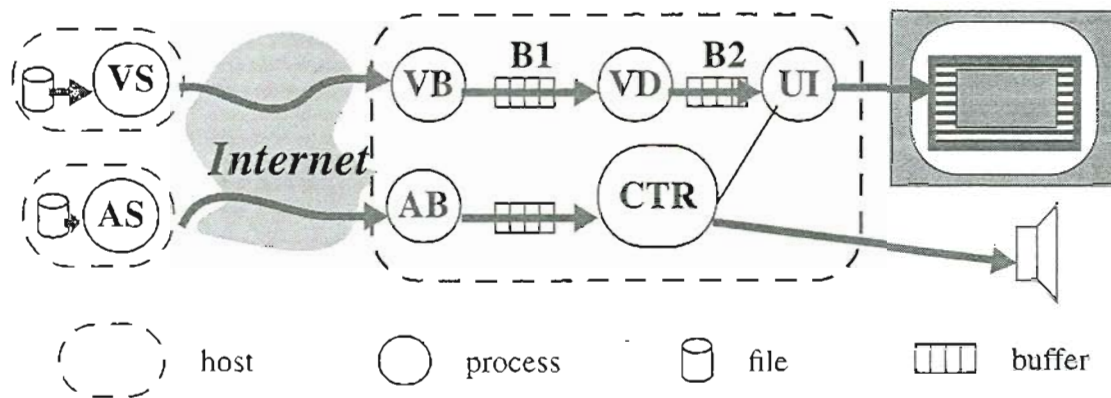
Figure 4: Detailed Architecture

Any stage in the video pipeline can drop frames if resources are insufficient,

- if packets are lost in the network due to congestion,
- if B1 overflows,
- if the decoder is too slow (insufficient CPU cycles), or
- if the frame is too late when it eventually arrives in B2.

Particularly at the decoder stage, MPEG's inter-frame dependencies require a sophisticated frame dropping policy. If an I-frame that is 10 msec late was dropped by VD, all subsequent frames referring to it can not be decoded, resulting in a gap of about 400 msec for a common frame pattern. Hence, it is only dropped if it is as much as 400 msec late. A similar problem exists for the frame rate control mentioned in Section 3.2. Figure 3 shows that not all frames are sent, in order to achieve a lower frame rate. To take frame dependencies into account, all B-frames are dropped before any P-frames and all P-frames before any I-frames. Figure 5 shows how send patterns are generated for given frame rates. For rewinding and fast forwarding only I-frames are used to avoid dealing with these inter-frame dependencies.

3.4 Feedback

As mentioned above, any overloaded pipeline stage can drop video frames. While VD takes MPEG frame dependencies into account, in the network and VB this dropping is done randomly and may result in high burstiness and jitter. Moreover, all resources used

```
frame
rate      send pattern

 1.5      ------------I-----------I-----------------------I----------
 2.5      I-----------I------------I-------------I-----------I----------
 5.0      I--P--------I--P--------I--P--------I--P--------I--P--------
10.0      I--P--P--P--I--P--P--P--I--P--P--P--I--P--P--P--I--P--P--P--
15.0      I--PB-P--PB-I--PB-P--PB-I--PB-P--PB-I--PB-P--PB-I--PB-P--PB-
20.0      I-BP-BP-BP-BI-BP-BP-BP-BI-BP-BP-BP-BI-BP-BP-BP-BI-BP-BP-BP-B
30.0      IBBPBBPBBPBBIBBPBBPBBPBBIBBPBBPBBPBBIBBPBBPBBPBBIBBPBBPBBPBB
```

Figure 5: MPEG frame send patterns

earlier in the pipeline for processing data that is dropped at a later pipeline stage are wasted. For instance, a frame that is dropped by the control process, because it arrives too late in B2, has already consumed disk bandwidth, memory, and CPU on the server, network bandwidth, as well as client memory and CPU cycles for decoding. A software feedback mechanism is used to adapt the rate at which the server sends frames to the rate at which frames are actually displayed. In this way, the player can adjust to dynamically changing resource availability without relying on any additional information about resources and bottlenecks. Figure 6 shows the mechanism: The raw display frame rate is passed through a low-pass filter in order to remove high-frequency noise. The control algorithm then adjusts the server frame rate when necessary.

F_{rd} raw display frame rate F_u user-specified frame-rate
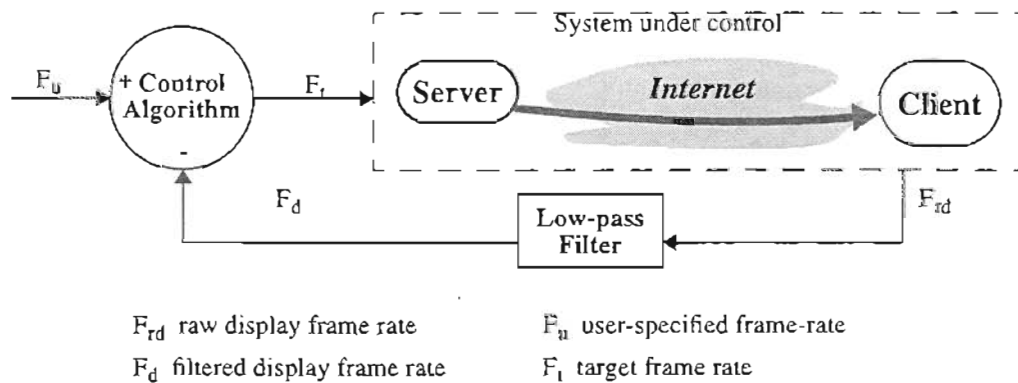F_d filtered display frame rate F_t target frame rate

Figure 6: QoS control Feedback [4]

A second feedback mechanism is used for synchronization between client and server clocks. The system clocks of both machines are not synchronized across the Inter-

net. A difference between the clock rates can make them drift apart, causing the client buffers eventually to become empty or overflow. A temporary skip or stall of either clock can change the buffer fill level permanently. Moreover, the fill level of the client's input buffer B1 should not be higher than needed for removing network delay jitter, because buffering reduces responsiveness to user controls. These problems are addressed by controlling the amount of time the server works ahead of the client in order to compensate for transmission delays. As shown in Figure 7, the current (raw) workahead time can be derived from the client's system clock and the time stamps of received packets. Transient noise is again removed by a low-pass filter before passing the data to the control algorithm. Simultaneously, the variation of the work-ahead time is monitored providing a measure of the current network jitter. This data again is used to adapt the target server workahead time: the more the server works ahead, the higher the client's buffer fill level, and the more jitter can be compensated for. For the audio pipeline a similar mechanism is used.
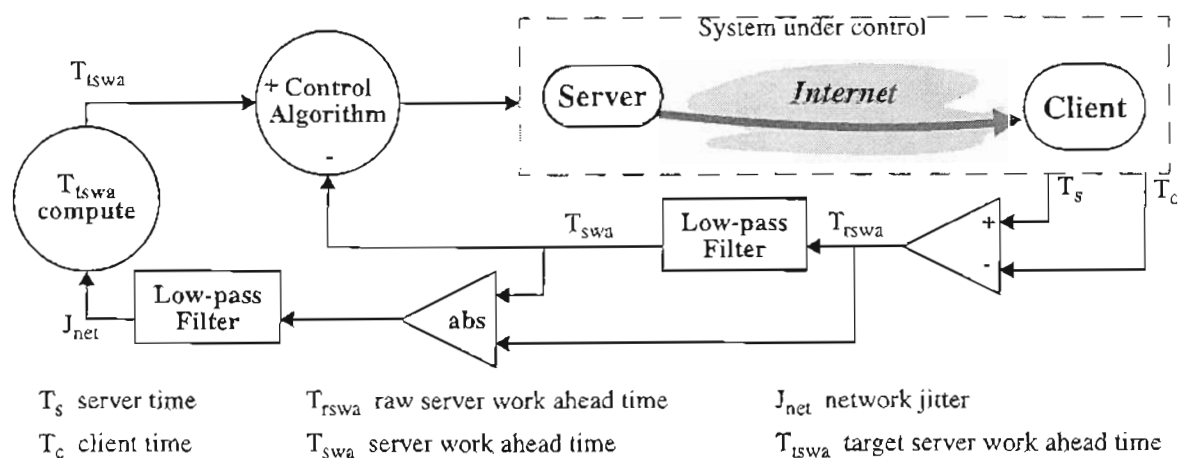
Figure 7: Synchronization Feedback [4]

3.5   Summary

This distributed player allows real-time retrieval of continuous media across the Internet. As a means of view and quality control, play speed and frame rate can be con-

trolled independently of each other. For video compression the MPEG standard is used. This format introduces inter-frame dependencies that need to be taken into account by frame rate control and frame dropping policies. Synchronization between client and server clock as well as dynamic quality adaptation is implemented by feedback mechanisms.

This player was used as the starting point for this thesis research. It has been extended and modified to add support for variable spatial dimension and complex content presentations.

Chapter 4

Design

4.1   Introduction

Cen's player provided the starting point for integrating QoS control in the spatial dimension and the capability of playing presentations composed of several streams. The first extension was support for multiple spatial resolutions. Initially, the size of the output image changed with the spatial resolution. Subsequently, these two parameters were decoupled to allow the user to choose an arbitrary size for the output image, and switch resolution independently of it. For composing presentations two main operations are needed: synchronizing parallel streams and concatenating streams. The implementation of these features required significant changes to the player's architecture. Moreover, for both operations, a means for prefetching had to be added to synchronize stream start events with the rest of the presentation. A simple description language is used for authoring these complex presentations.

The following sections describe the architectural design changes required to support these features. The last section of this chapter explains some implementation details such as the structure of C++ objects and buffer management used.

4.2   Multiple Resolutions

To adapt quality better in accordance with the preferences of the user, control over spatial resolution was added in addition to control over temporal resolution. Since MPEG-1 does not provide several quality levels for one source file, separate files for different res-

olutions are used, as shown in Figure 8. A configuration file at the server side is used to specify that a set of files are actually the same movie at different resolutions. The video server checks that the lengths and frame patterns of all files are the same. When changing the resolution, the server simply switches to another source file. To locate the position of a particular frame in a file, index tables are used. They are generated once and stored on disk for future accesses. These tables allow random access to frames while searching for a position in the video clip, fast forwarding, and skipping frames while playing. When a video with multiple resolutions is initialized, the indices of all files are loaded into main memory. This approach allows a quick switch by simply using another file and another index for locating a particular frame.
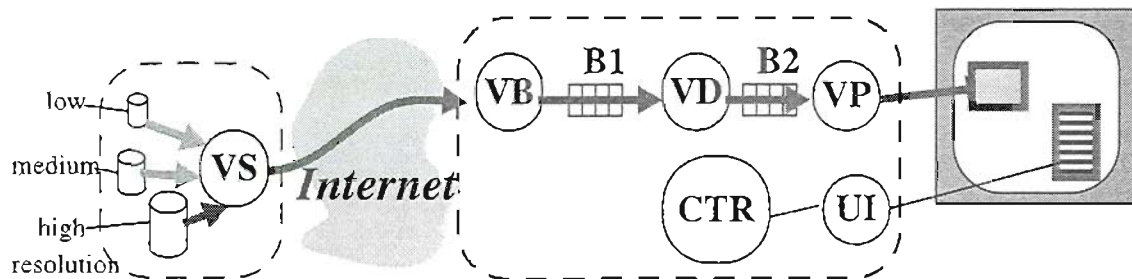
Figure 8: Multiple Resolutions

To update the state of the MPEG decoder, the MPEG header with the new resolution information is sent. Since this new header can be lost in the network, the decoder could try to apply an obsolete header to new data. Hence, the old header has to be invalidated causing the new one to be requested from the server and resent if lost. To do so, the client needs to be informed of the resolution switch. Using the control connection for this task would require some kind of synchronization between the control and data stream: It is necessary to find out at what frame a change notification takes effect. To avoid this problem, size information is sent over the data connection with every video frame.

Currently, the resolution is manually controlled by the user through buttons that can increase or decrease it, or a pop-up menu to select a particular resolution directly. Future versions should include this QoS dimension in the player's automatic adaptation mechanism.

Storing the same content several times at the server is a very simple way of providing scaling without support by the compression algorithm. Providing files with four reso-

lution steps is possible with about twice as much disk space as having the highest resolution only. The same approach can also be used for other ways of scaling. Some MPEG decoders, for instance, allow encoding at a specified bandwidth. Hence, providing MPEG files encoded for different bandwidths also works with the implemented technique.

4.3   Variable Image Size

The player allows the size of the video output to be different from the resolution in the source file: the images can be scaled. This feature enables the user to control better how he wants to see a presentation. Image size is a second variable view parameter in addition to play speed. Moreover, only because the current resolution is independent of the output window size, changing the resolution really is a means of quality control. Without scaling, a resolution switch would alter video image size, changing not only the quality but the entire appearance of the presentation. Figure 9 illustrates several combinations. As shown in Table 3, view and quality can be controlled orthogonally in the spatial as well as in the temporal dimension.

high resolution   low resolution   low resolution
large image   large image   small image
no scaling   scaling   no scaling
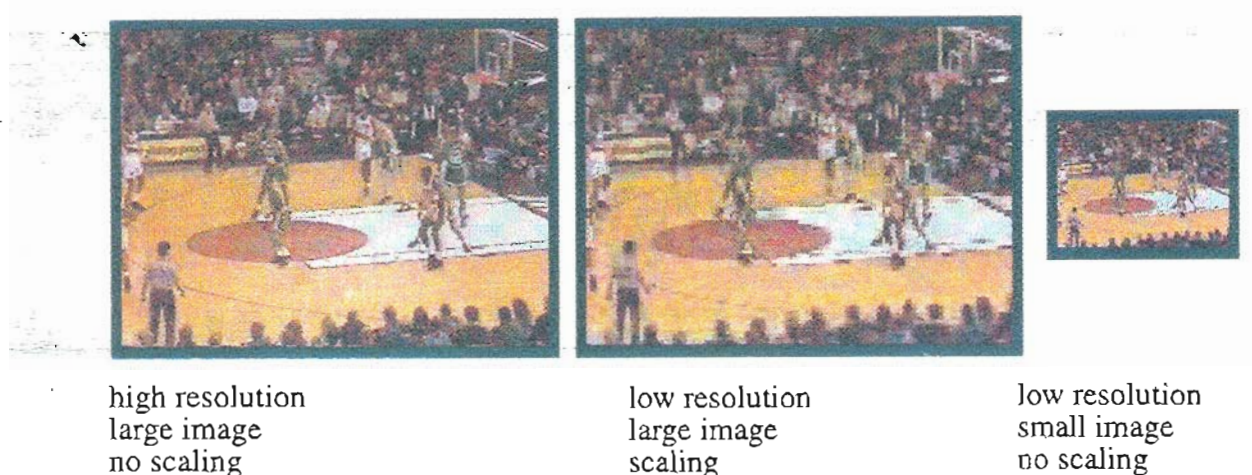
Figure 9: Resolution and Image Size

Table 3: Orthogonality of View and Quality

	view	quality
temporal	play speed	frame rate
spatial	image size	resolution

The scaling is currently optimized neither for performance nor for image quality. The implementation is simply a linear projection. This scaling step is added in the VD process between the decoding and dithering of the image. Note that dithering consumes about as much CPU capacity as decoding. The resource consumption for dithering depends only on the size of the output image and is not reduced by choosing a lower quality, that is, a lower resolution.

The dithering algorithms as well as B2 containing the final images must be able to change image size dynamically. As a convenient way for the user to control the image size, videos are displayed in windows on their own. The window size can simply be changed using the window manager. The player monitors these changes and adjusts the scaling accordingly. Figure 10 shows that video output and control panel are in separate windows, in contrast to the old version.
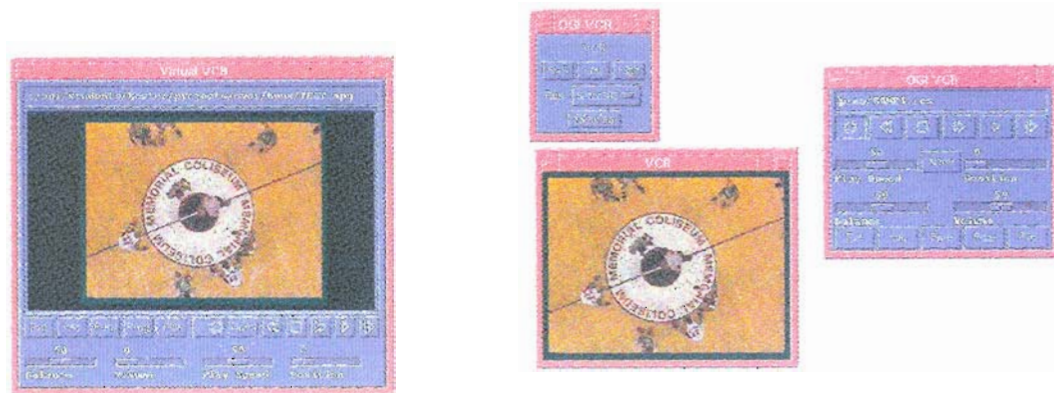
Previously: One Window

Now: Resizable Video Window and per-stream resolution control menu

Figure 10: Interface for Variable Image Size

4.4   Multiple Video Streams

For complex presentations, synchronized playback of several streams is necessary. To implement this feature, several instances of the entire video pipeline must be created, as Figure 11 demonstrates. This capability required a major restructuring of the previous architecture shown in Figure 12. The UI process needed to be split into a part for manag-
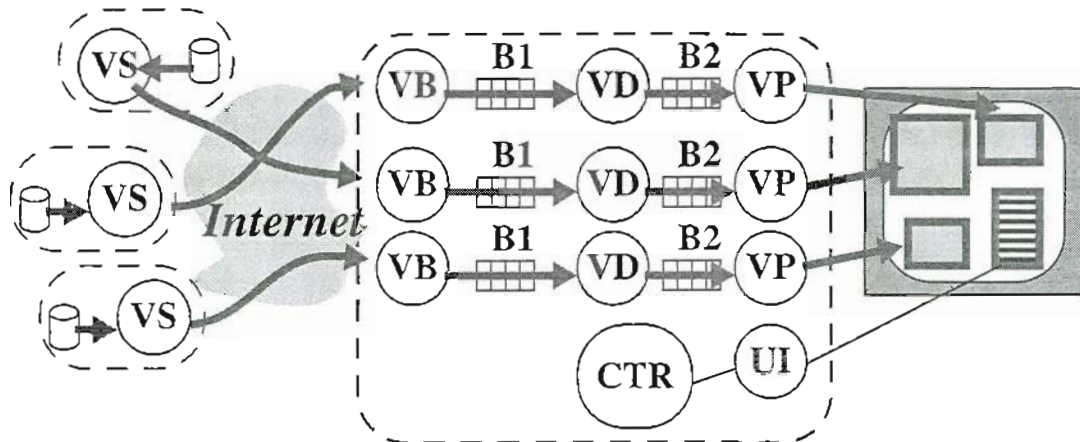
Figure 11: Multiple Video Streams
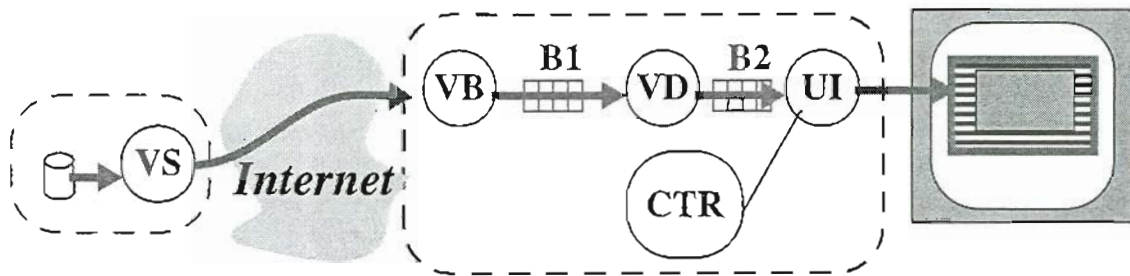
Figure 12: Previous Architecture

ing the control panel and a per-stream part (VP) handling the video output window. The VB and VD processes as well as the buffers B1 and B2 were made replicable. Global variables had to be changed to variables local to the pipeline module. Since 'fork' copies the entire address space, and hence even global variables, replication of VB and VD could have taken advantage of this property. To provide a better program structure, however, all processes have been encapsulated in modules, as will be discussed in Section 4.9. More-

over, the removal of global variables allows replicating the modules even if threads are used rather than processes. This property should facilitate porting the player to systems that support multithreading. Restructuring the status variables in shared memory and the CTR process was a major task. All variables and pieces of code had to be separated in a per-stream part, that is replicated for every pipeline, and a part controlling the overall presentation. CTR has been completely reengineered using a uniform stream abstraction for video as well as audio streams.

The timing control had to be changed, too. Time used to be measured in terms of frame serial numbers and driven by a periodic timer interrupt. When several video streams with potentially different frame rates are supported, this approach is not possible any more. The player's notion of time is now measured in terms of milliseconds from presentation start at normal play speed. The mapping between this logical time and the real time measured by the system clock depends on the actual play speed. Timed events such as displaying of a frame or playing a block of audio samples are managed by an alarm clock module using event list scheduling. The clock blocks the control process until the next event needs to be processed. The time-out parameter of the Unix system call select is used to implement this timer. Commonly the accuracy of this mechanism is 10 milliseconds. In comparison, the time between display events at a rate of 30 frames per second is 33 milliseconds. Hence, at high frame rates the granularity of this timer is a significant source of jitter. More system support is needed to provide a higher quality.

There is no need to explicitly deal with synchronization while playing. All frame display events are controlled by the system clock of the client machine. Hence, the synchronization error between two streams is at most the sum of their timing errors. The server clocks are kept synchronized with the client by the feedback mechanism described in Section 3.4. Figure 13 shows a presentation with three synchronized video streams showing a basketball game from different camera angles.

Playing of several audio streams is currently not supported. In contrast to video with an output window for each stream, multiple audio streams would need to be merged and put to a single output device.

Figure 13: Presentation with Three Video Streams

4.5 Start-up latency

A problem introduced by distributed complex content is compensating for different start-up latencies. When a stream starts playing, a command is sent to its server through the TCP control connection. The server then starts sending the data at the specified frame rate. If there is only one stream, there is no problem: The client timer can simply be started, when the first data packet arrives.

client timer		1	2	3	4	5	6	7	8	9	10	11
received	Latency	1	2	3	4	5	6	7	8	9	10	11
output		1	2	3	4	5	6	7	8	9	10	11

If there are several streams, they may have different latencies and this procedure does not work any more. Because late video frames are still displayed unless there are more recent ones, an ostrich approach can be considered: Ignore the problem, send the

commands and start the timer at the same time.

client timer	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
received	Latency				1	2	3	4	5	6	7	8	9	10	11
output					1	2	3	4	5	6	7	8	9	10	11
received	Latency			1	2	3	4	5	6	7	8	9	10	11	12	13
output				1	2	3	4	5	6	7	8	9	10	11	12	13
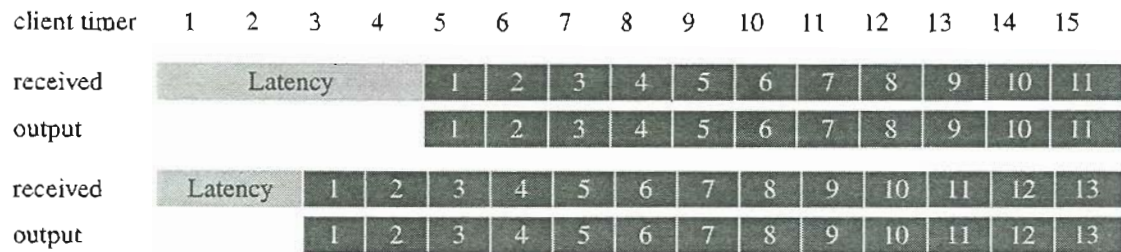
After a few seconds the synchronization feedback mechanisms would speed up the server clocks to catch up with the client timer. However, the streams would be unsynchronized during that time. Moreover, audio behaves differently. Samples need to be continuo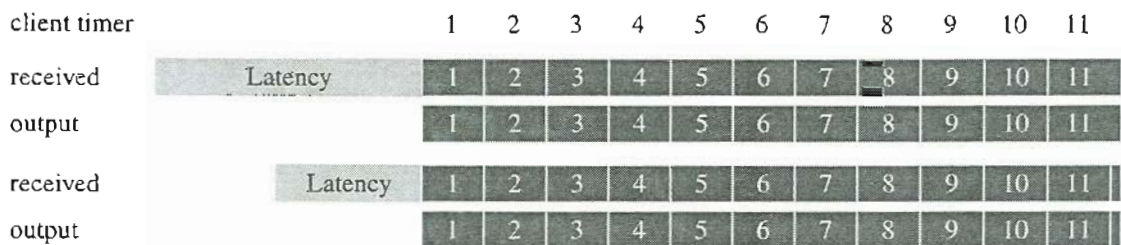usly transferred to the audio device. If the data is not there yet, silence samples are played. Whereas late video frames still can be used, late audio samples cannot. That means that with the approach described above audio samples would be played only once the feedback synchronizes the streams. Because that can take a few seconds, the approach is unacceptable.

Ideally, the sending of commands and starting the timer should be scheduled so that the streams really start together.

client timer		1	2	3	4	5	6	7	8	9	10	11
received	Latency	1	2	3	4	5	6	7	8	9	10	11
output		1	2	3	4	5	6	7	8	9	10	11
received	Latency	1	2	3	4	5	6	7	8	9	10	11
output		1	2	3	4	5	6	7	8	9	10	11

This approach, however, requires knowing the latency in advance. Obtaining a reliable latency estimate by pinging the server or similar mechanisms is inappropriate, because in a congested network the variations are too big. This effect probably is caused by the use of a reliable protocol for the commands: It is not predictable how often the command packet needs to be resent. A protocol other than TCP may back-off less and reduce the problem, but not solve it.

Another method has been used for synchronizing audio and video in the previous architecture and has been adopted for the general case of multiple streams. The commands

are sent at once and the timer is started when data of all streams has arrived:

client timer		1	2	3	4	5	6	7	8	9	10	11
received	Latency	1	2	3	4	5	6	7	8	9	10	11
output		1	2	3	4	5	6	7	8	9	10	11
received	Latency 1 2	3	4	5	6	7	8	9	10	11	12	13
output		1	2	3	4	5	6	7	8	9	10	11

During the time before the client timer is started the buffers of streams with low latency are filled. If buffers overflow, data will be lost. This effect is unlikely, however, because the synchronization feedback is active during that time. The client clock does not advance yet whereas arrival of new packets indicates the progress of server time. Hence, the feedback mechanism slows down the server and tries to synchronize both clocks. While the server clock is not halted completely, the mechanism reduces chances of a buffer overflow significantly and works well in practice.

4.6   Concatenation

Besides playing synchronized streams, concatenating streams is the other basic feature needed for complex presentations. The component streams may be retrieved from different servers with potentially different latencies to the client.

Two simple solutions for switching between streams are possible: The first is simply to reinitialize the video pipeline with a connection to the server of the second stream. This procedure, however, takes too long and would result in a large gap in the output stream. The second possibility is replicating the entire pipeline for concatenated streams, too. At the time for the switch, the second pipeline is ready and can start quickly. This approach is too wasteful of resources. Because some presentations may concatenate many short streams, many pipelines would be needed, each of them consisting of three processes and two large buffers. Moreover, a way of joining the pipelines to one output window would have to be found. The approach implemented is somewhere between these two alternatives. For all streams to be concatenated the connection to the server, including the VB process, is maintained during the entire presentation. Having all connections ready

allows quick switching between them. The rest of the pipeline is not replicated. All VB processes write to the same B1 buffer, as shown in Figure 14.
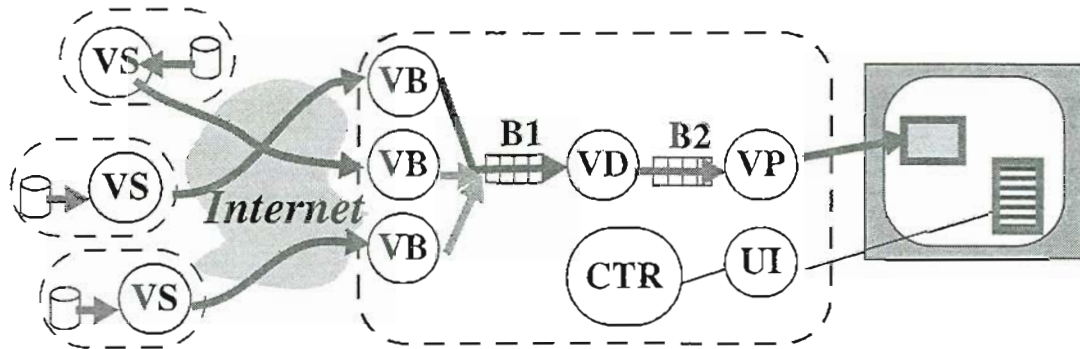

Figure 14: Concatenation

The access of several VBs to B1 needs to be synchronized. An initial solution was to use signals to deactivate and reactivate the VB processes. Implementing the merging in B1 turned out to be a more robust solution. The buffer simply accepts only data from the active VB process and ignores (drops) data from inactive ones. This mechanism can be implemented transparently to the VB processes.

The timing of the switches between streams is driven by the CTR process. An additional control structure maintains the concatenation-related data and sends the appropriate commands to the respective servers. Not only play commands, but also fast forward, rewind, or position must be directed to the right stream.

The video pipeline has to deal with data of several streams, which may be encoded with different characteristics such as different resolutions and frame rates. The end of the pipeline may still process frames of the first stream whereas the beginning already deals with the second stream. Hence, stream dependent information cannot simply be maintained in global status variables, but is now associated with every frame in the pipeline. In this way, every stage can detect a change of parameters and adapt its local state accordingly at the right time.

Concatenation of audio streams has been implemented in the same way.

4.7 Prefetching

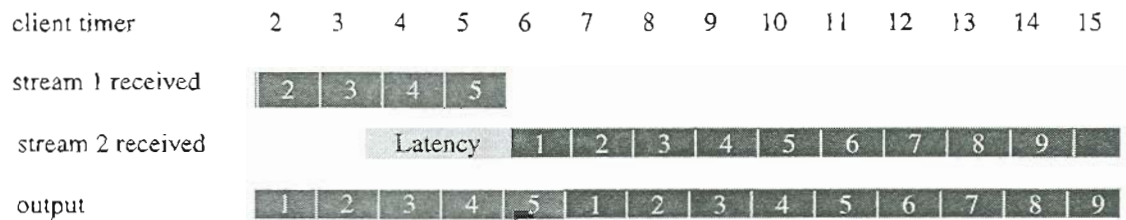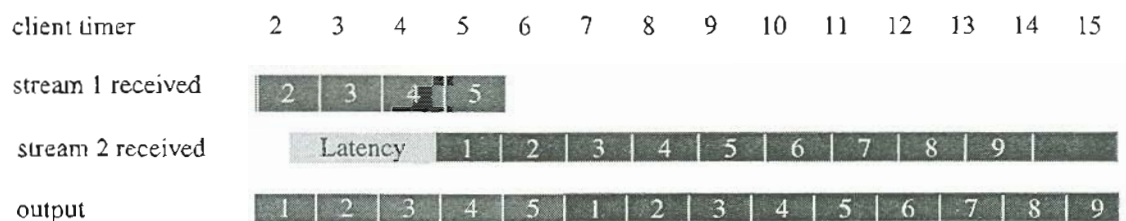Concatenation introduced a timing problem, too. Although the connections of all streams are kept open and ready, the latency after sending the play command remains, resulting in a gap in the output stream:

client timer	2	3	4	5	6	7	8	9	10	11	12	13	14	15
stream 1 received														
stream 2 received														
output														
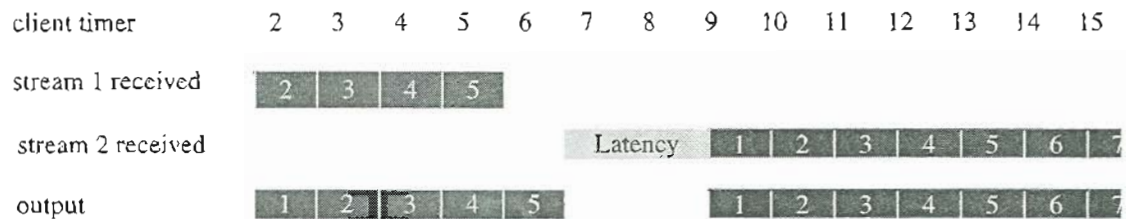
Ideally, again, it would be best to schedule the sending of the command ahead by precisely the latency period in order to achieve a smooth transition. Notice that – while playing – the data arrives some time before it is displayed, providing a certain buffer fill level that allows the player to eliminate delay jitter.

client timer	2	3	4	5	6	7	8	9	10	11	12	13	14	15
stream 1 received														
stream 2 received														
output														

Because the precise latency is not known, it is not possible to schedule the start command the exact amount of time ahead. The same problem occurred for synchronized start-up. In practice, sending the start command a constant time before the actual switch yields useful results.

client timer	2	3	4	5	6	7	8	9	10	11	12	13	14	15
stream 1 received														
stream 2 received														
output														

This way of prefetching produced two additional requirements for the architecture. It must be possible to communicate with servers other than those which are currently play-ing. Moreover, as can be seen in the previous figure, it must be possible to receive data of

several streams concurrently. In the current implementation, the feedback mechanism is not active while prefetching, causing prefetched data always to be sent at full frame rate.

Again, the control structures in CTR had to be split in one part containing all information related to a particular connection and another part representing the state of the rest of the pipeline. The per-connection parts can communicate with their server independently of other stages of the pipeline allowing to send prefetched play commands while a different stream is being played. Buffer B1 has become a double buffer: It has a list of frames for the active and one for the prefetched stream as shown in Figure 15. Frames are added to the respective list according to the VB process that is delivering them. Obsolete frames from a VB process that is neither active nor prefetching are simply dropped.
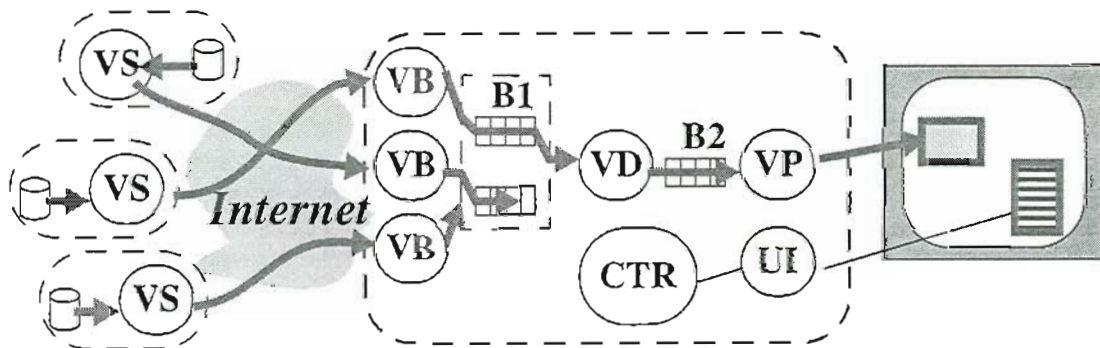

Figure 15: Prefetching

This approach allows the prefetching of only one stream at a time and does not work well if clips are shorter than the latency. For the current implementation, however, considerable changes of the control and buffer structure would be necessary to allow prefetching an arbitrary number of streams in parallel.

4.8   Complex Presentations

In addition to multiple and concatenated streams, two other features for complex presentation were implemented: gaps and clips. Before, after, or between concatenated streams, gaps can be added. This functionality is integrated in the CTR module that controls concatenation. The gap is handled as a dummy stream. The other feature is playing clips. That is, only part of a stream can be selected, which is very useful in authoring pre-

presentation =	(audio | video)+	A presentation consists of one or more audio and video streams that are played simultaneously. Currently, only one audio stream is possible.
video =	`video` **stream** [`position` *x-pos y-pos*]	optional window position
audio =	`audio` **stream**	
stream =	([gap] file)+	A stream is a sequence of clips. Optionally, there may be gaps between clips.
gap =	`gap` *msec*	
file =	`file` *server-host file* [`offset` *offset_msec*] [`length` *length_msec*] [`speed` *speed_percent*]	offset in the source file length of the clip default speed relative to recording speed

Figure 16: Presentation Description Language

sentations. A clip is described by its length and its offset from the beginning of the source file, both in terms of time. A possible extension could allow selecting also a spatial subpart of the images.

The presentations are defined by a simple language that is described in Figure 16. The author must compose the presentation as a synchronization of a concatenation of streams. This restriction is inconvenient. Support of an arbitrary mixture of synchronization and concatenation operators would be helpful to authors, but is beyond the scope of this thesis.

4.9   Implementation

The complexity of the presentations supported is reflected in the internal structure of the player. For every component of a presentation, separate processes and control structures are needed. Splitting the program in a per-stream, a per-pipeline, and an overall control part required some reengineering of the previous architecture.

The processes are encapsulated in modules. Wrapping them up in C++ objects helped to define clear interfaces. Global variables in shared memory are now logically part of modules and can be accessed only through module functions. "Static" variables are integrated in objects. This restructuring has several advantages: The program has become more readable and complexity has been broken down, facilitating maintenance and further enhancements. It is now possible to create several instances of modules as needed. The use of threads rather than processes becomes possible.
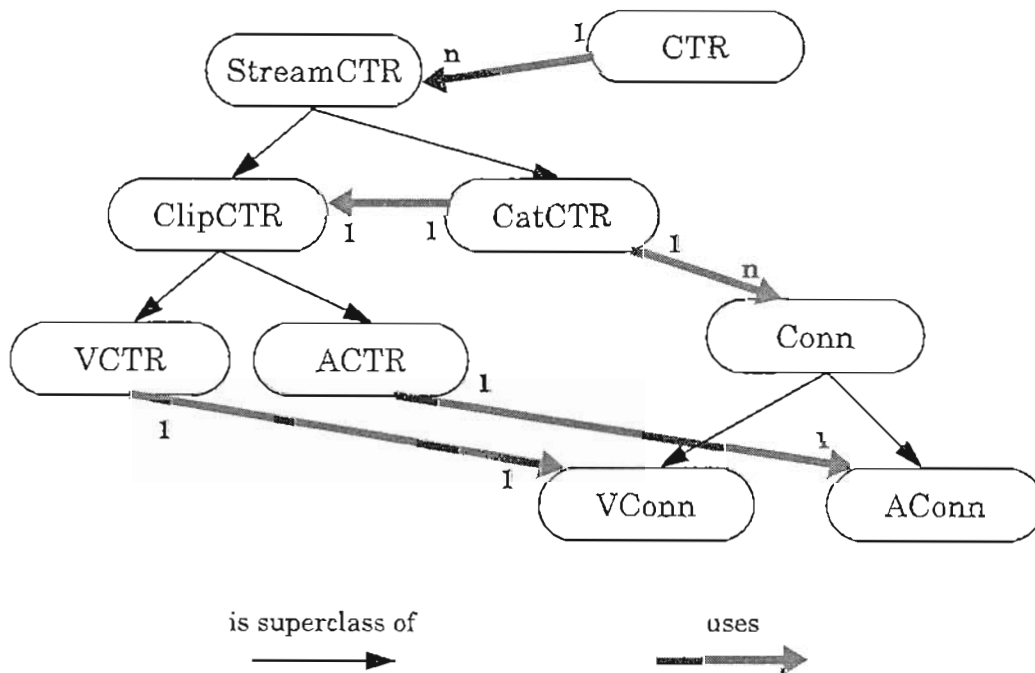
Figure 17: Object Hierarchy

The control process CTR has been restructured using C++ objects, as shown in Figure 17. The overall control CTR handles a list of pipelines that process synchronized streams. A pipeline can be a simple video (VCTR) or audio (ACTR) pipeline, or a concatenation of streams (CatCTR). VCTR or ACTR have exactly one active connection (VConn or AConn) plugged into the pipeline. CatCTR is a wrapper around a pipeline of either type maintaining in addition a list of connections, which can be plugged into the actual pipeline to switch between concatenated streams. Having a uniform stream (ClipCTR) and connection (Conn) abstraction made it possible to deal with a concatenation of video

streams in the same way as with a concatenation of audio streams.

Synchronizing the access to buffers has been complicated by stream switches. Waiting for a full or empty buffer slot was implemented by counting semaphores. This method worked because there was exactly one process putting frames into the buffer and exactly one taking frames out. When switching between concatenated streams the control process now causes obsolete data to be dropped, acting as an additional consumer and hence interfering with this simple access pattern. Implementing the buffers as monitors with condition variables for 'full' and 'empty' solved this problem. Moreover for B1 a generic memory management mechanism had to be implemented to allow parallel access of several VB processes and variable frame sizes.

4.10 Summary

This player provides control over temporal as well as spatial resolution. The use of several files encoded with different quality allows spatial scaling even if the compression algorithm does not support extracting data at different quality levels from one source file. In addition to the feedback mechanism that automatically adjusts the frame rate to the current resource availability, the user can now choose between multiple resolutions to control quality. Having more than one variable quality dimension provides several ways of adapting to the current resource availability. Choosing the one best for the user maximizes achievable quality.

Independently of these quality parameters, the user can change the view by choosing play speed and image sizes. For decoupling the resolution at the source file and the output image size a scaling step was added between decoding and dithering of video frames.

Complex presentations are supported. It is possible to concatenate streams as well as play several streams simultaneously, with streams being retrieved from potentially different servers. For concatenation it is necessary to switch quickly between succeeding streams. To achieve this goal, several sources are dynamically re-connected to a continuous media pipeline. Several pipelines are needed for playing synchronized streams. A modular program structure facilitates separating per-stream processes and control struc-

tures from stream-independent parts of the program. A shared timer in the client keeps the pipelines synchronized with each other, and a feedback mechanism keeps the servers synchronized with the client. Varying network latencies make it hard to synchronize the start time of streams with the rest of a presentation. The presentation description is used to initiate prefetching that hides the start-up delay.

Chapter 5

Performance

Control over the spatial QoS dimension in addition to the temporal dimension is supposed to improve QoS management. Several experiments show the effect of both parameters on key resources. All experiments are done with one Basketball video at different resolutions. Because MPEG bandwidth and decoding times depend on the content of the video, the quantitative results may differ for other clips. The given examples, however, are sufficient to provide some insight to the effects and problems involved.

5.1 Resource Consumption

We want to control resource consumption by the quality parameters of resolution and frame rate. In our environment and player configuration, client CPU and network bandwidth have shown to be scarce resources, and other bottlenecks have rarely been observed. Hence, this experiment investigates the effect of both quality parameters on these two resources.

For measuring CPU consumption it is sufficient to look at the VD process, since all computationally intensive tasks, that is decoding, scaling, and dithering, are done there. A one minute video clip has been run for a set of frame rates and three spatial resolutions. The image has not been scaled, but was displayed at the size corresponding to the resolution. For each run the CPU time consumed by VD has been queried using the clock function of Unix. Figure 18 displays the results. As the CPU time approaches 60 seconds, which is the real time available for displaying a one-minute clip, higher frame rates become impossible, because frames would be dropped. To complete the graph even so, the
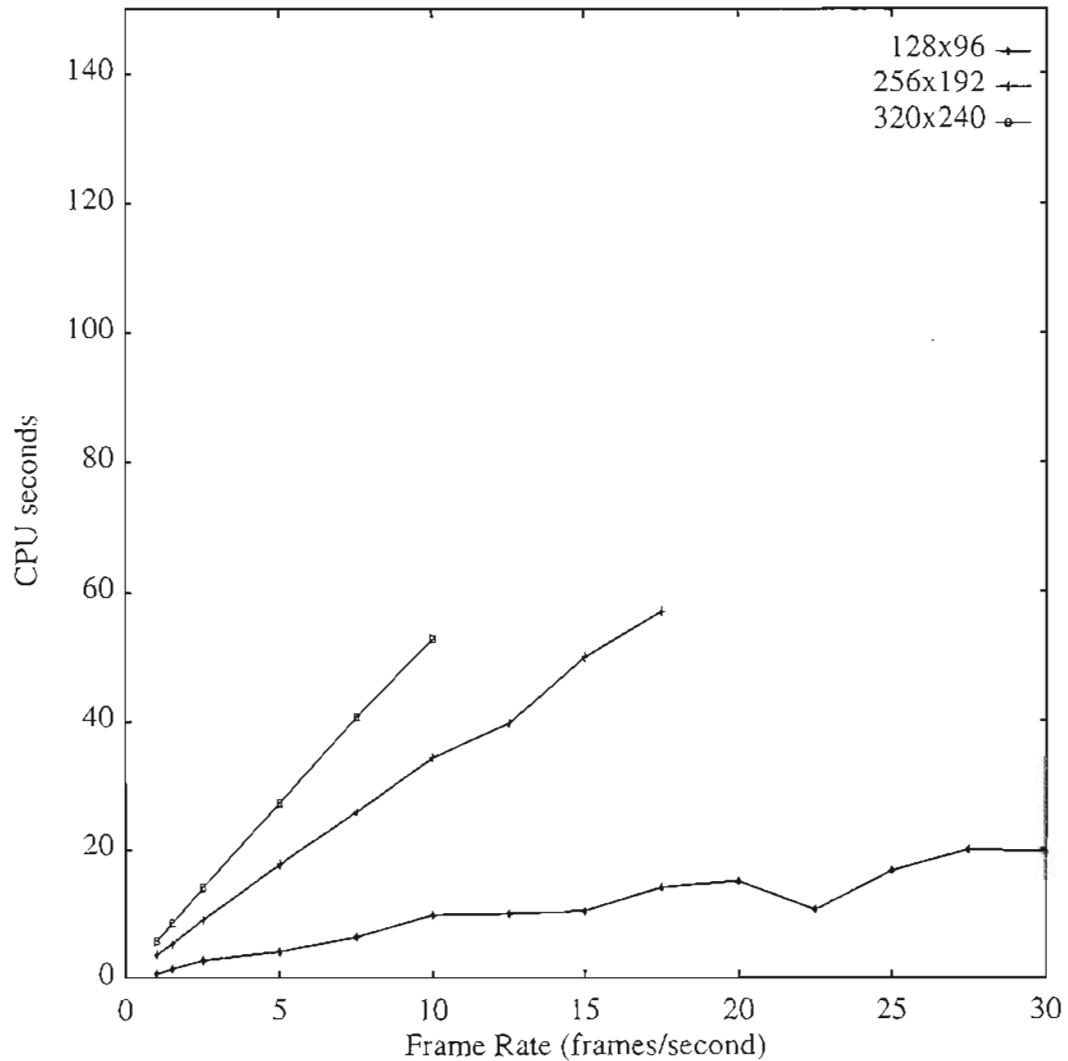
Figure 18: CPU consumption per frame rate and resolution at normal play speed

same experiment has been run at 20% of the normal play speed. It is shown in Figure 19. To facilitate comparison of the graphs the frame rate is given relative to normal play speed. Actually, it was 0.5 to 6 frames per second in the slow speed experiment.

The figures show that frame rate and resolution can be used to control resource usage trade-offs. With 30% of the CPU for instance, you can get low resolution at 30 frames per second, medium resolution at 6 frames per second, or high resolution at 3 frames per second.

In the same experiment the average bandwidth for the one minute clips has been measured, as shown in Figure 20. Also for network capacity, resolution and frame rate can
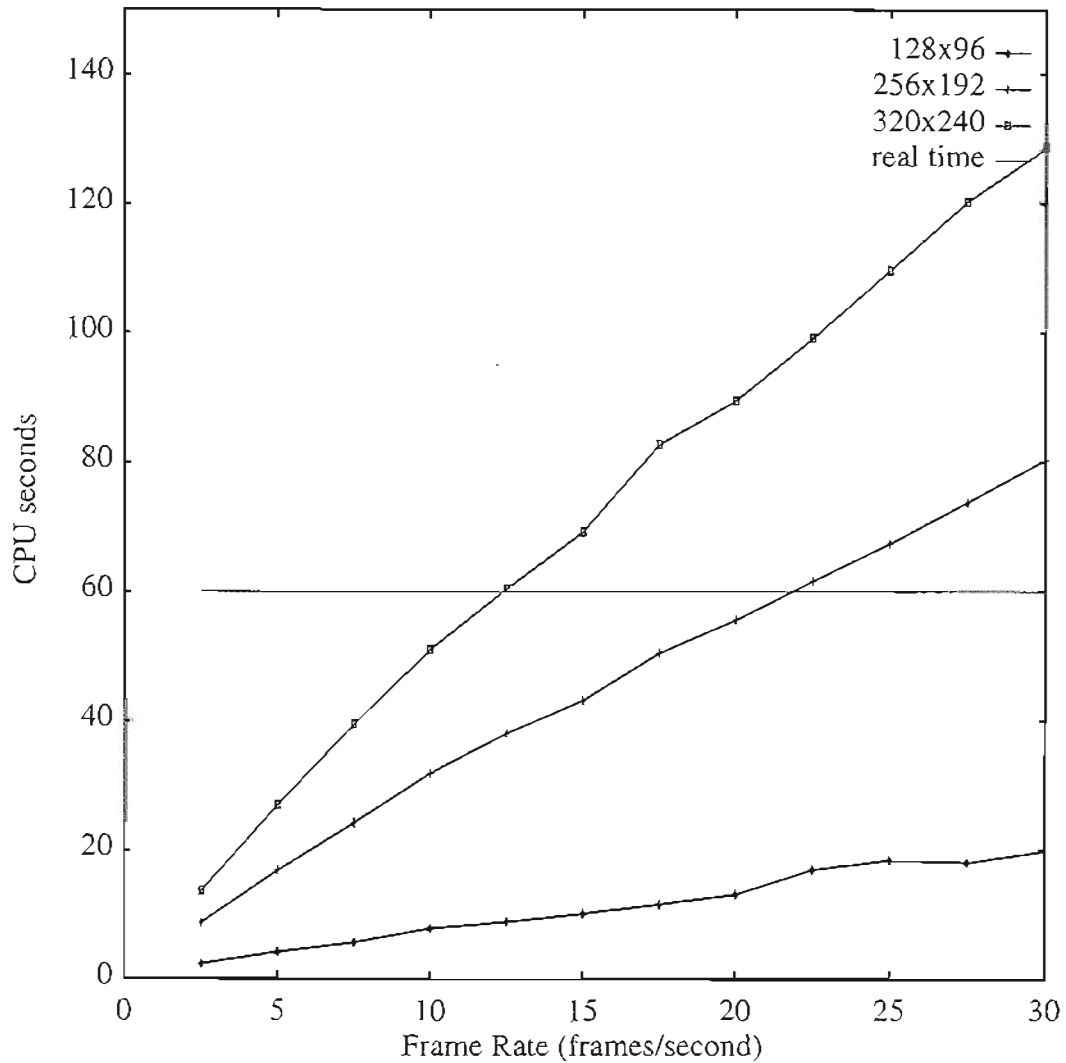
Figure 19: CPU consumption per frame rate and resolution at slow play speed

be traded for each other. At 300 kb per second, for instance, 4 frames per second can be played at high resolution, 7.5 frames per second at medium resolution, or full frame rate at low resolution with only about half of the bandwidth used. High bandwidth, however, could not even be utilized, because the CPU becomes the bottleneck. Note that the bandwidth graphs consist of three linear parts. These sections are caused by the send pattern of MPEG frames: At 1, 1.5, and 2.5 frames per second only I-frames are sent, at 5, 7.5, and 10 frames per second P-frames are added, and B-frames are included for higher rates.[1]

---

1. The send pattern mechanism is explained in Section 3.3. The example in Figure 5 shows the frame pattern used in this experiment.
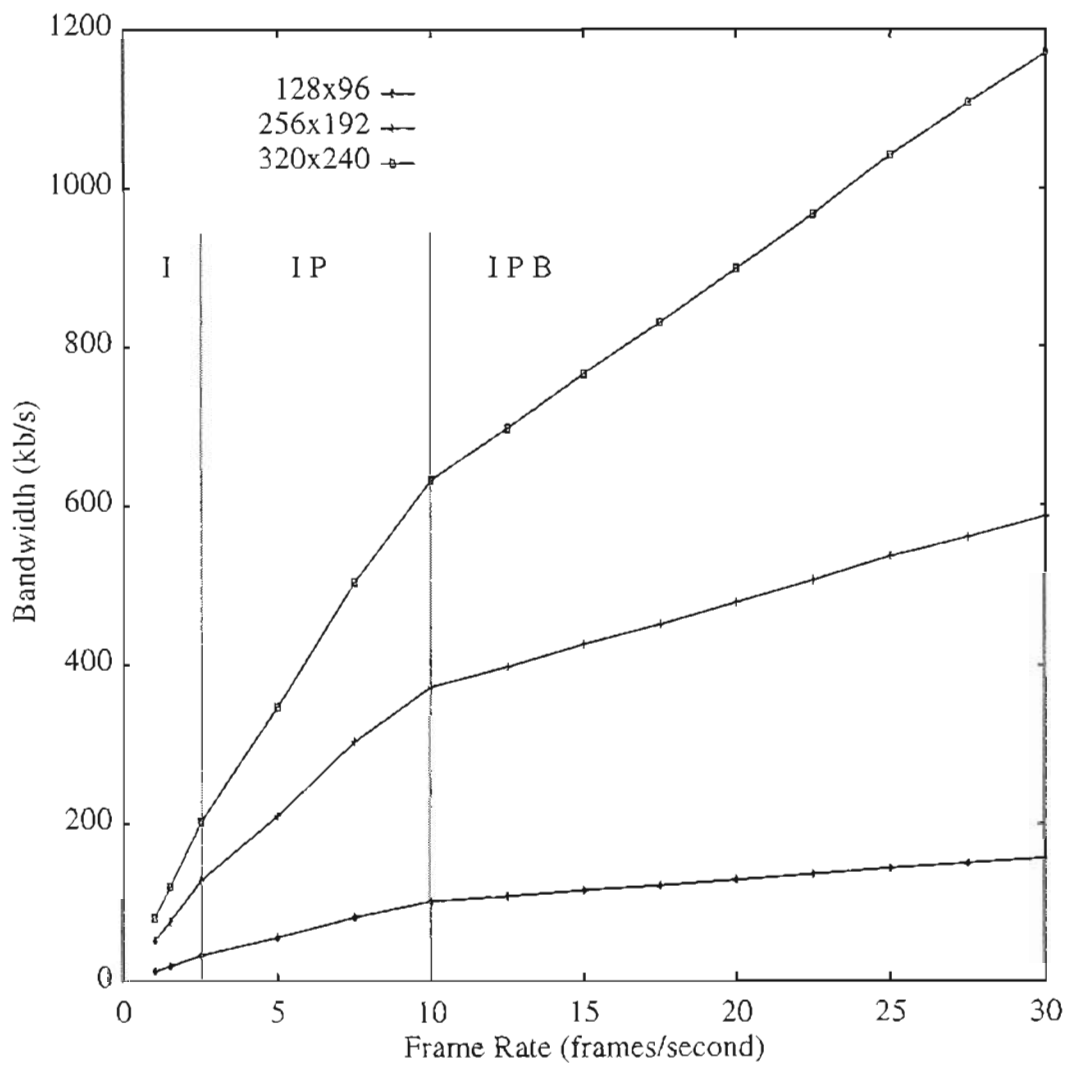
Figure 20: Bandwidth per frame rate and resolution

5.2 Adaptation with two Quality Dimensions

The feedback mechanism provides automatic fine-grained adaptation of the frame rate. Resolution changes are rather coarse grained and are currently controlled by the user. The following graphs show how these two mechanisms interact. Each figure represents one presentation. The rate at which frames are sent is shown as well as the rate at which frame are displayed. Because time information about display events contains a lot of noise, the average time between the last 25 display events is used to generate the latter graph. At the times indicated in the figures the resolution was changed by the user. The feedback then adapted the frame rate to the new resource requirements.
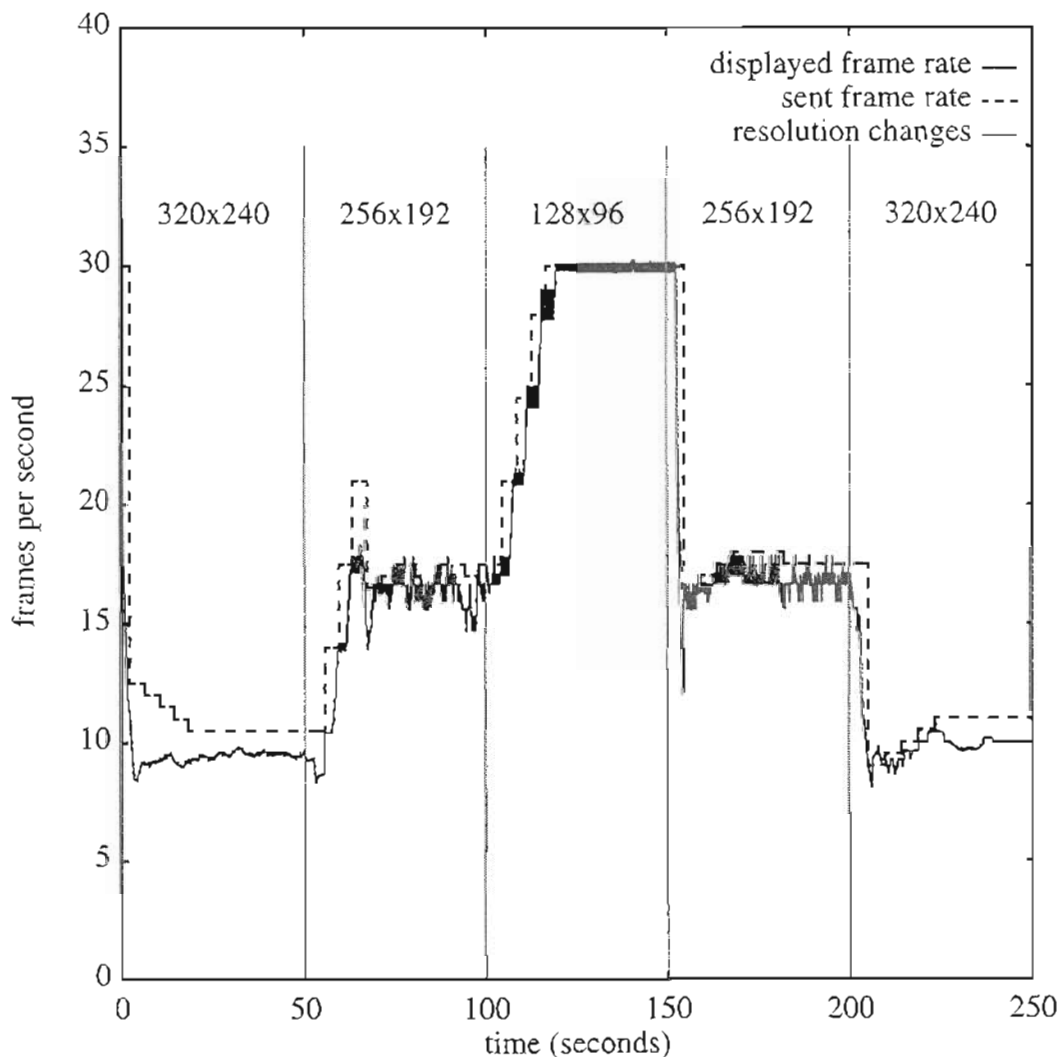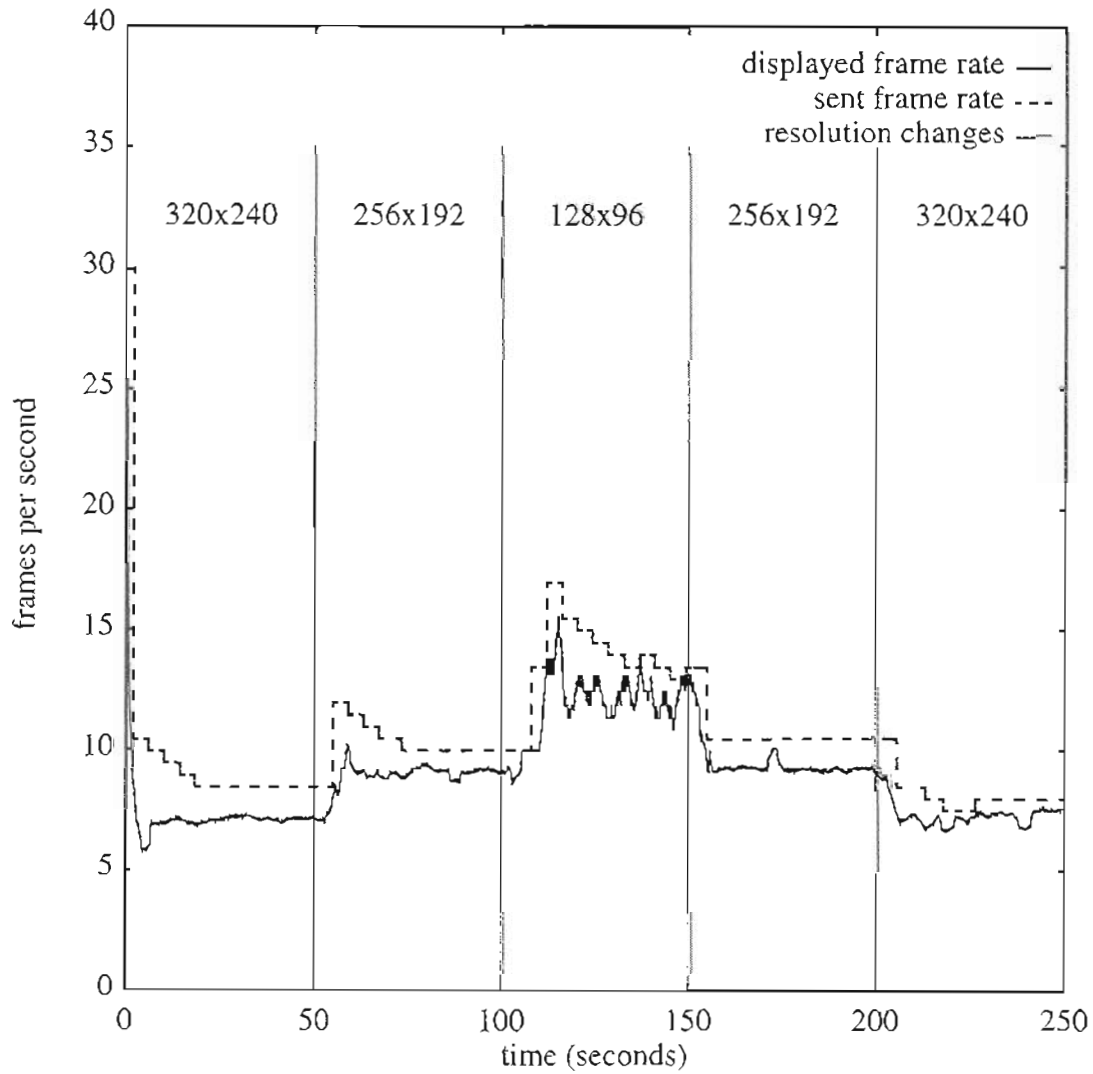

Figure 21: Adaptation, CPU scarce, without scaling

Figure 22: Adaptation, CPU scarce, with scaling

The presentations in Figures 21 and 22 have been retrieved from a server on the same machine as the client. In this case, the client CPU becomes the bottleneck. In Figure 21, the video image is not scaled, that is the output image size is changed together with the resolution. The graphs clearly show that each resolution allows a different frame rate, and that the feedback quickly adjusts to the new conditions. The feedback mechanism usually adapts in .5 frames-per-second steps. After resolution switches it is temporarily sped up to simply drop to the current display frame rate or to explore higher rates until the pipeline becomes overloaded.

Ideally a change of quality should not affect the appearance of the presentation,

that is in this case, the image size should remain the same. This feature has been activated for Figure 22. The decoded images have been scaled to a size of 320x240 regardless of the resolution. The adaptation works is in this case, too. The client CPU resources required for scaling and rendering, however, reduce the possible frame rates significantly. Not only quality, but also view has a considerable impact on CPU consumption. Section 5.3 provides a closer look at this problem.
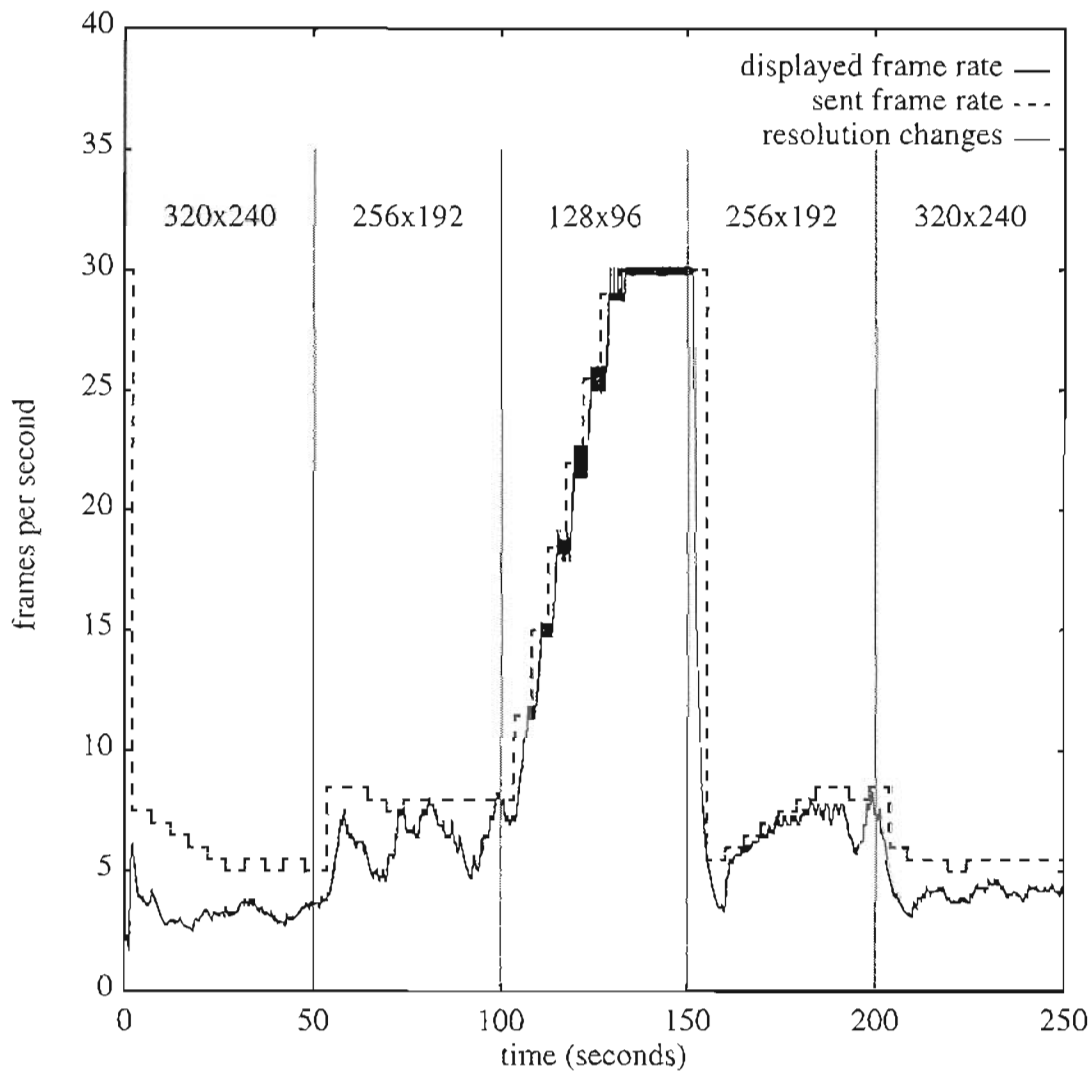

Figure 23: Adaptation, bandwidth limited to 300 kb/s

In Figure 23 network bandwidth is the bottleneck. The output image is not scaled, because it would make the CPU the scarcest resource in most cases. To simulate the network bottleneck, the server has still been run locally but a bandwidth limiter has been

used. This limiter is implemented at the server. It monitors the bandwidth at which the server sends. Every time a packet is to be sent, the limiter checks whether the current bandwidth exceeds 300 kb/s. If that is the case, the packet is simply dropped with a 50% chance instead of being sent, simulating network contention. Dropped packets cause the current bandwidth to decrease. If it drops below 300 kb/s the next packet will be sent again without a chance of being dropped. Figure 24 shows the resulting relation of sent to
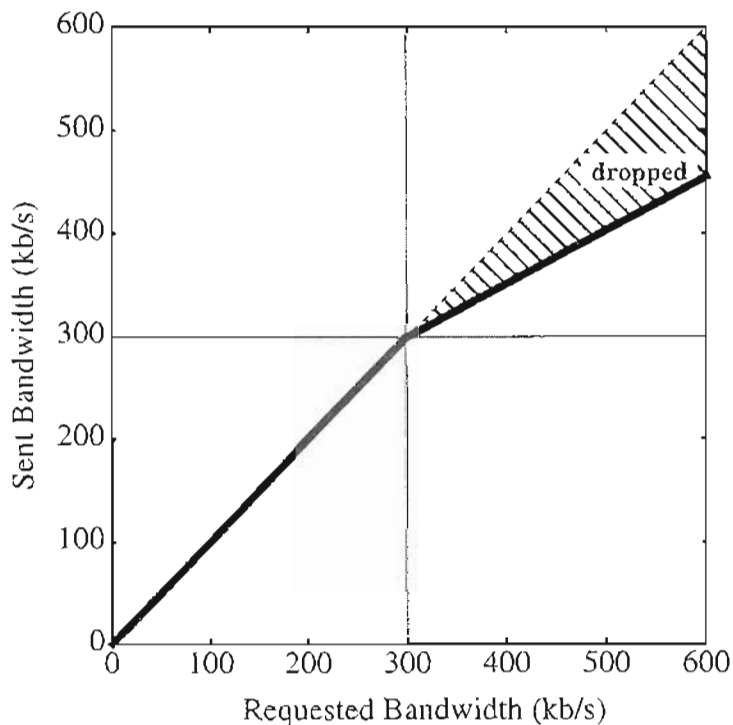

Figure 24: Simulated Network Contention

requested bandwidth. The low resolution is not affected by this limitation, but the frame rates for medium and high resolution are reduced. Whereas a CPU bottleneck causes frames to be dropped by the decoder considering MPEG frame dependencies, the bandwidth limiter as well as a real congested network does not take them into account. If an I-frame is dropped, all dependent frames cannot be displayed either. This behavior causes a higher burstiness.

The same experiment has been run over the Internet. The remote server has been located at the University of Kaiserslautern, Germany, 20 hops away from the client at OGI. On this link out-of-order delivery of UDP packets can occur, which requires the player to reorder them. Moreover, frames may have to be split into several UDP packets. If
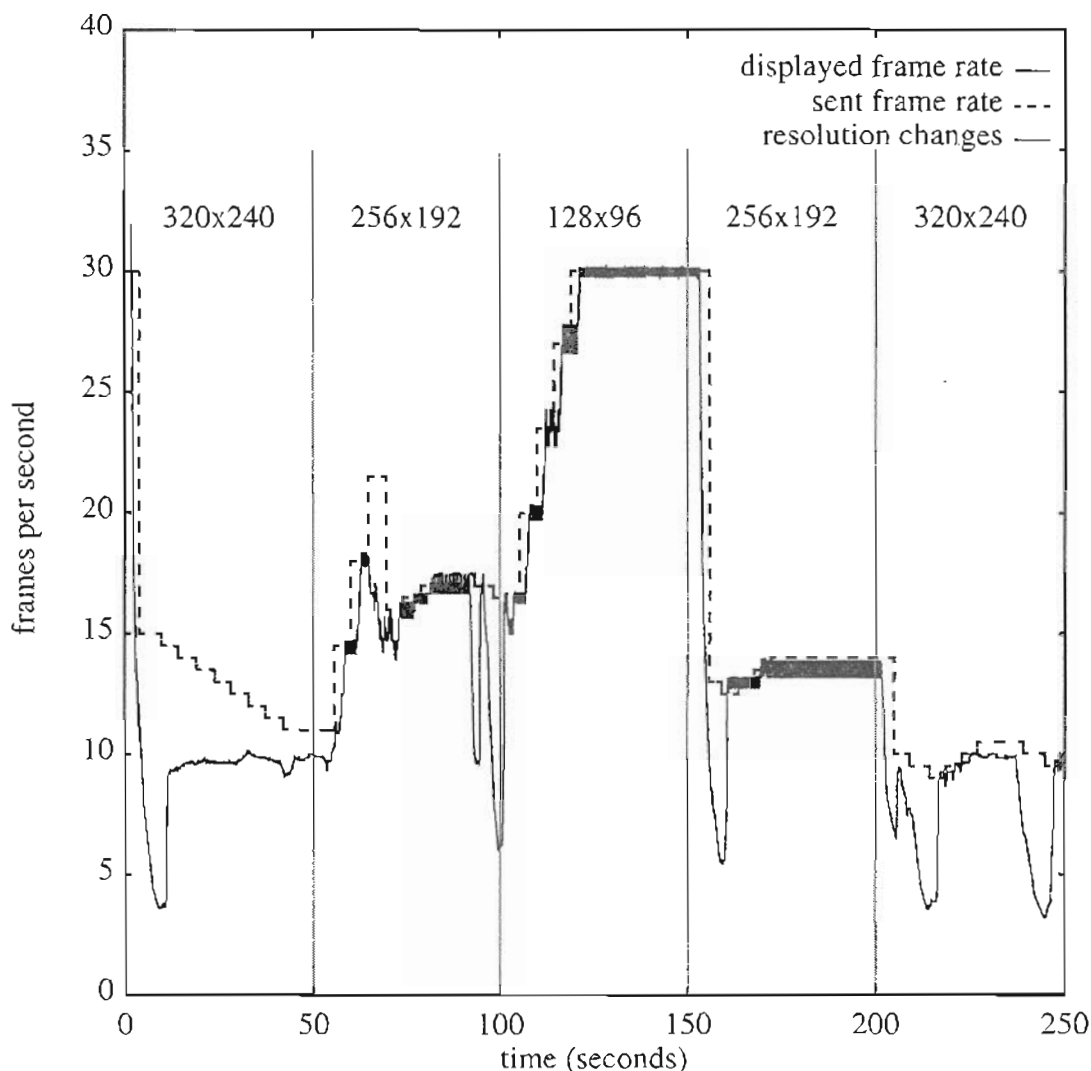
Figure 25: Adaptation, remote server in Germany, uncongested network

one of them is lost in the network, the entire frame must be dropped. This dependency makes larger frames more sensitive to packet loss. I-frames, for instance, are more likely to be dropped than B-frames, but the latter cannot be decoded without their reference frames, anyway. Hence, the player performs quite badly if the network is overloaded, and it depends heavily on the adaptation to prevent this situation. Network congestion and player performance vary a lot. Hence, the three graphs presented are not necessarily representative, but they are examples that demonstrate some effects that occur when playing over the Internet.

The graph in Figure 25 shows an almost ideal performance. This presentation was
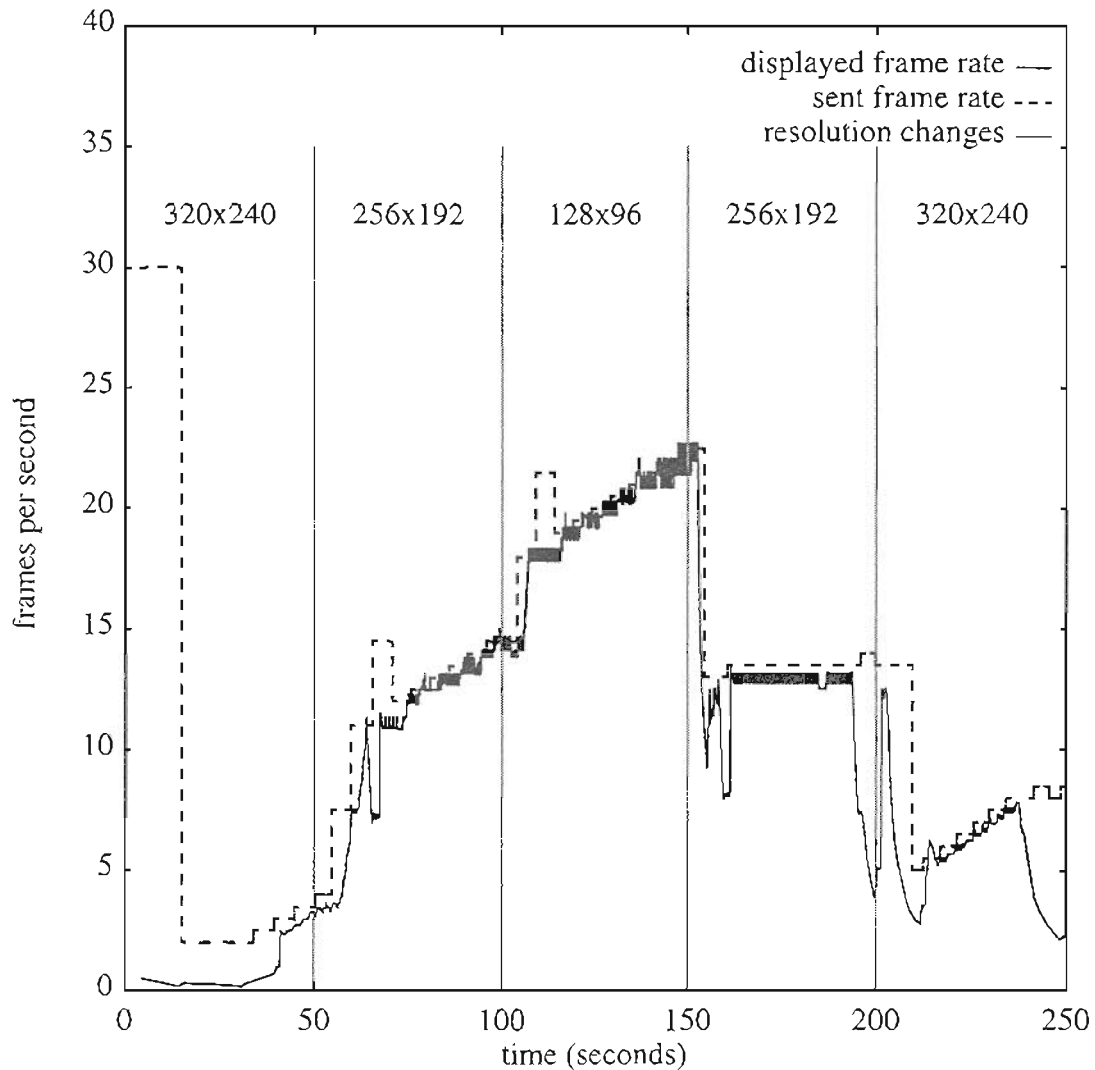
Figure 26: Adaptation, remote server in Germany, congested network

recorded on a Saturday morning when there was little load on the network. Only some temporary frame rate drops differ from the local case. The graphs in Figure 26 and 27 were measured on a weekday on a congested network. Figure 26 shows that the start-up at the high resolution causes serious problems. The player stalls for as much as 40 seconds. Several factors contribute to this behavior:

- The server starts sending with the full frame rate causing an overload situation.

- At start-up, the player prefetches the beginning of the video clip. During this prefetching, the frame rate feedback is not active, and frames are still sent at 30 frames per second.

- Since I-frames consist of many network packets and are more likely to be dropped, it

can take some time before the first frame can be decoded at all.

- The feedback is data driven. Whenever a frame is displayed, the display frame rate is compared to the server frame rate. If there is no frame displayed at all, however, the feedback will not take any action.

- The player's simple packet reordering mechanism can delay the delivery of frames and introduce high burstiness.
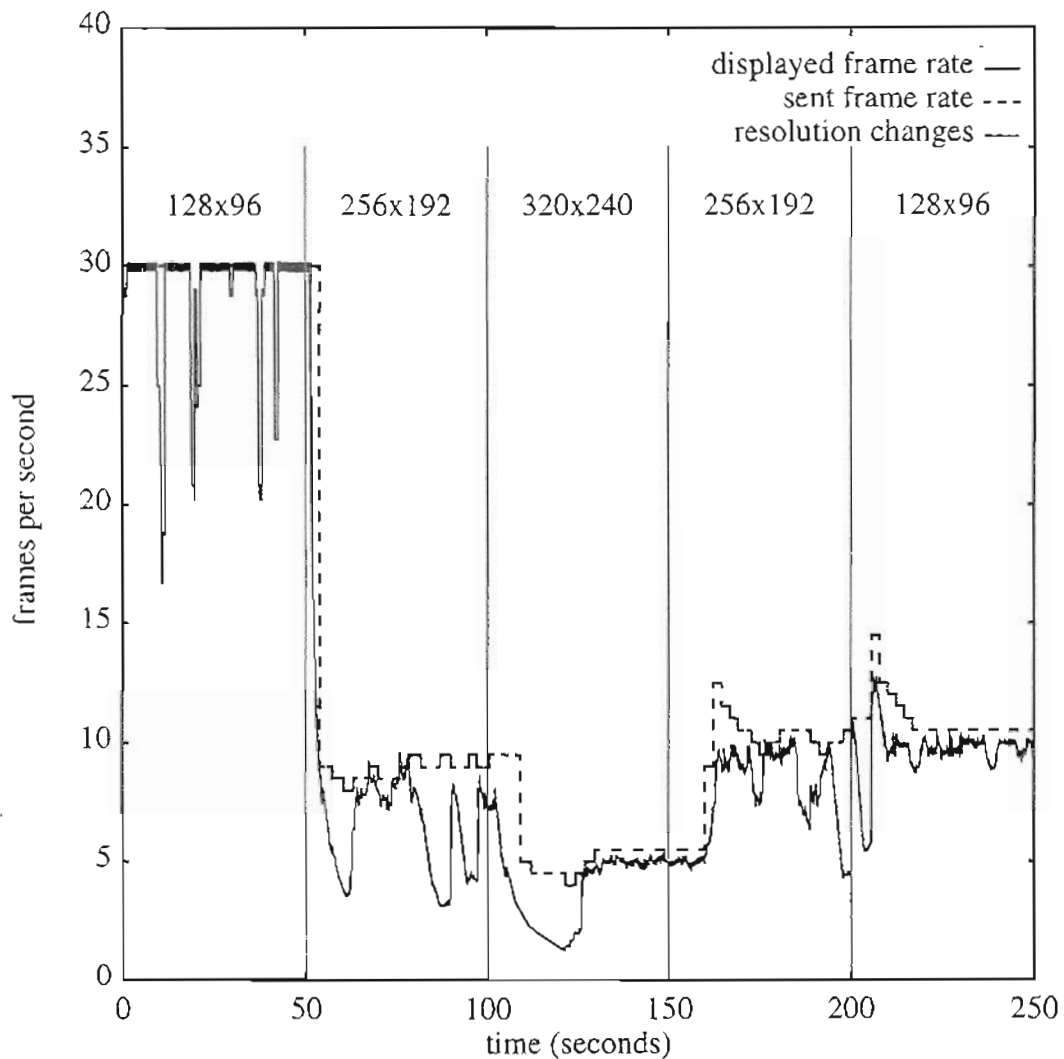
Figure 27: Adaptation, remote server in Germany, congested network

For the low resolution 30 frames per second are usually possible even if the network is busy. After switching to the low resolution in Figure 26, the mechanism erroneously assumes to be already in a stable state after the change and tries only slowly to

increase the frame rate. In Figure 27, the high rate is achieved after start-up, but after the resolution change at 200 seconds, adaptation fails entirely, that is, the frame rate does not increase at all after switching to the low resolution. This problem shows that the reaction to resolution changes needs to be better integrated with the feedback. Another factor may be that the available bandwidth is not static even if the competing traffic is not changing. The player's UDP packets may cause competing TCP traffic to back off increasing its own bandwidth share.

5.3　View Impact on CPU Consumption

As Figure 21 has shown, CPU consumption depends not only on resolution but also on image size. To measure this effect, a 10 second clip has been played with different image sizes. The play speed was 20% to allow all frames to be displayed without overloading the CPU.
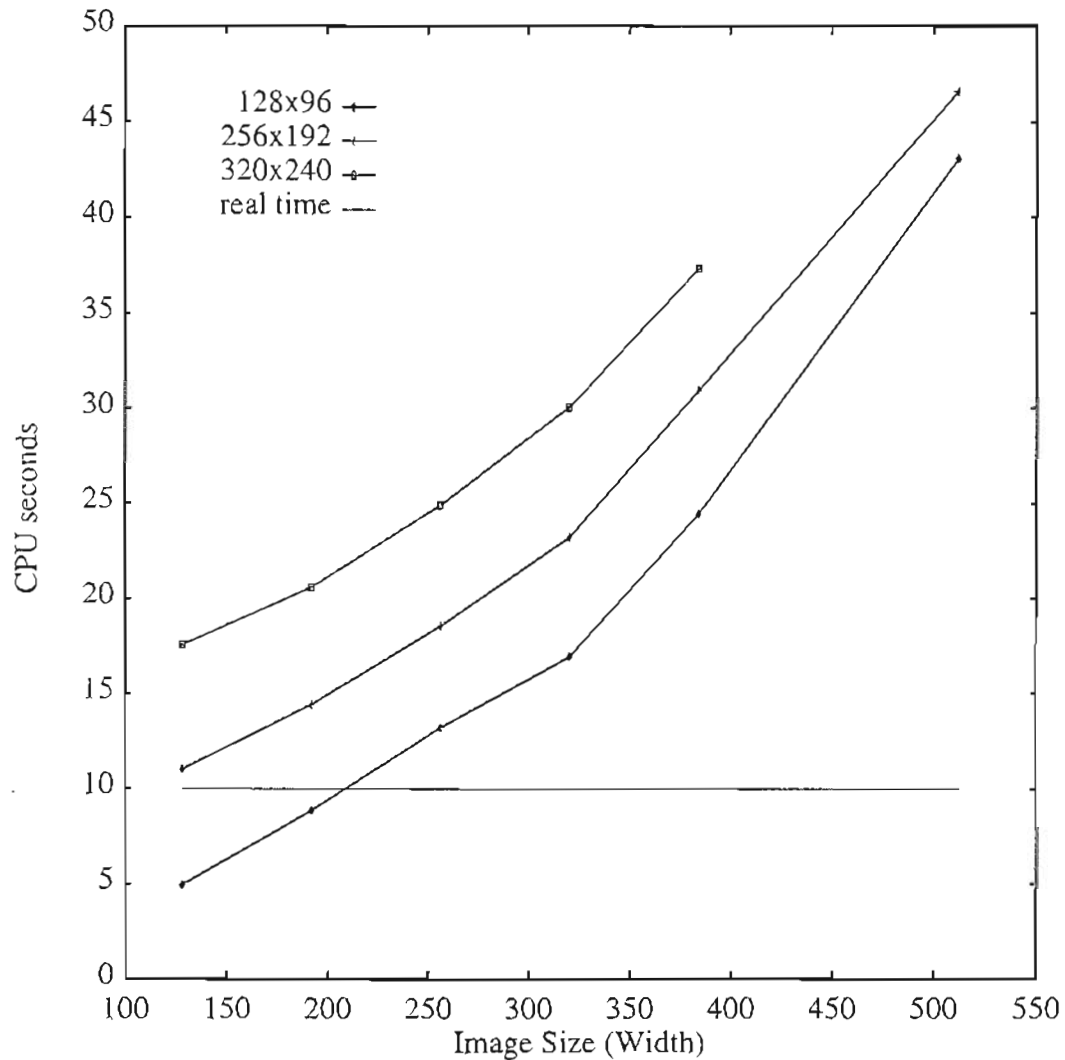
Figure 28: CPU consumption and image size for different resolutions

In Figure 28 the CPU times are shown for three resolutions and six image sizes of the clip. The x-axis is labeled with the image width, with the height being always 3/4 of the width. The CPU consumption increases significantly with the image size. At normal
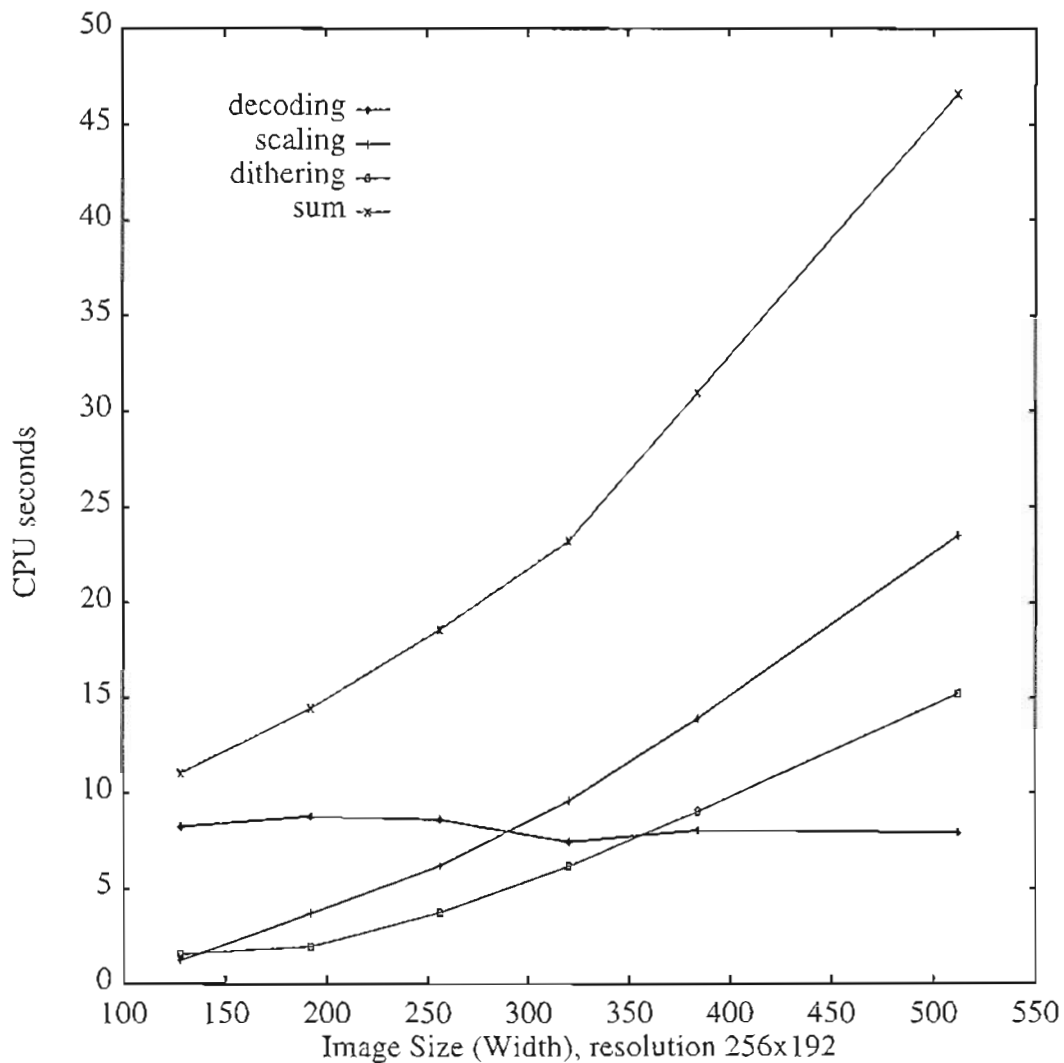
Figure 29: CPU consumption and image size for different tasks

play speed, only the lowest resolution with the two smallest image sizes could be displayed at 30 frames per second. The largest image with the highest frame rate could not even be managed at 20% speed, which is at five times the CPU capacity.

The CPU consumption for the medium resolution is split up in Figure 29. The decoding time depends on the resolution, whereas scaling and dithering increase with the image size. Although recent optimizations have shown that the scaling algorithm can be made about three times faster, scaling and dithering have considerable costs. The view of a presentation – not only its quality – has an effect on resource consumption.

Chapter 6

Future Work

6.1   Automatic Adaptation of Resolution

Currently image resolution can only be controlled manually by the user whereas the feedback mechanism adapts only the frame rate automatically. It is desirable to include all variable QoS dimensions in an automatic adaptation mechanism and to assess the trade-offs between them based on an error model. To do so, resolution switching needs to be integrated with the frame-rate-control feedback. Adding control over additional QoS dimensions such as color depth would allow us to provide better quality to the user in adapting to resource availability.

6.2   The Cost of Quality

A multimedia system that controls several QoS dimensions and uses several resources needs to have some way of relating QoS and resource requirements. An adaptive application needs to find the configuration with the best quality given the resources available, whereas for resource reservation it is desirable to find the configuration with the least resource requirements for a specified quality. Either problem requires a comparison of configurations with respect to the quality they provide as well as the amount of resources they consume. The value of a presentation needs to be related to its cost.

Staehli [31] uses his QoS model and error interpretation to evaluate the quality. In his player he uses simple heuristics to estimate the effect of QoS dimensions on resource consumption. One heuristic is that every dimension contributes equally to resource con-

sumption and another that resource requirements are higher for presentations with higher quality. To implement an advanced automatic adaptation policy it is necessary to find out if these heuristics are appropriate for a distributed player and if better ones can be found.

6.3   Error Model Extension

The error model presented by Staehli assumes a uniform quality for all streams. In general however, different parts of a presentation may have different quality requirements as mentioned in the examples of Section 1.2. Moreover, in a distributed player the resources available for different streams may vary a lot, resulting in errors varying among streams even if the quality specifications are uniform. Hence, the error model needs to be extended to include per-stream QoS specifications. The quality of a presentation can be calculated as some function of the quality values of all streams, perhaps a mean.

A potential problem is starvation. For instance, if there are two streams and frame rate is the only variable QoS dimension, displaying a frame of stream A could require fewer resources than a frame of stream B. Using a mean as the quality measure of the entire presentation in this scenario, playing stream A with 30 frames per second and stream B with none at all would provide better quality than playing both streams with 10 frames per second, which is likely to be more desirable. Including uniformity in the over-all quality measure may solve this problem.

6.4   Feedback for Multiple Streams

Currently the adaptation is controlled by per-stream feedback mechanisms that work independently of each other. This behavior may result in a random resource distribution among streams. An error model extended for multiple stream specifications would provide a desirable quality ratio between streams based on user specifications. A coordinated adaptation mechanism should aim at achieving this ratio. However, different resource availability may require different of adaptations. Let us assume we have two streams with stream A being twice as important as stream B. If both streams come from

the same source, that is they share all resources, the adaptation should achieve a 2:1 quality ratio. However, if stream A comes from a remote server over a low-bandwidth link and B is local, it does not make sense to needlessly degrade stream B's quality in order to keep it half as good as stream A's. The adaptation should rather present stream A with the best possible quality and spend the rest of the CPU on stream B. In general, this decision requires knowledge about how much of the resources is shared between streams. For the network, this information is difficult to obtain. It would be necessary to find out what fraction of the connections is shared between two remote servers. It is desirable to find a feedback solution that can yield useful results despite these problems.

6.5   Better Feedback Adjustment to Events in the Player

The feedback mechanism adjusts dynamically to resource availability, adapting to changes caused by other users that share resources as well as by the player itself. In general, this adaptation is done carefully and rather slowly to avoid oscillation. Hence, adaptation to a substantial change can take several seconds. This performance could be improved for the events caused by the player itself, such as change of resolution or play speed, switch between concatenated streams as well as end of a stream or start of an additional one. In complex presentations a variety of such events occur frequently. The presentation description can give the feedback mechanism additional information for a faster reaction, for instance by speeding up the adaptation rate temporarily or by using heuristics about the effect of the change.

6.6   Improved Presentation Authoring

The current restriction on composing presentations should be relaxed. Only synchronization of concatenation of streams is currently supported. An arbitrary combination and nesting of these operators is desirable. Staehli [31] proposed a normalization algorithm that could be used to transform a description to the currently supported form.

Moreover, it would be useful to include presentations in other presentations.

Accessing content descriptions from remote sites rather than having to have them at the client would be another improvement. Supporting synchronized playing and mixing of several audio streams would add a helpful feature, too.

6.7   Integration with Admission Testing

For use in an environment that supports admission testing, an appropriate interface between player and admission tester must be provided. As mentioned in Section 6.2, a way of estimating the resource consumption of a clip is necessary. This QoS mapping would allow calculating resource-level requirements such as network bandwidth or CPU consumption from the content information and the quality parameters of a stream. The varying needs of a complex content presentation make this problem more complicated. The resource requirements can be submitted to an admission tester.

Most user commands, such as 'Play' or 'Rewind', would invoke the admission tester. 'Stop' could release the resources, whereas 'Pause' could keep them, guaranteeing that resumption of the presentation is possible. Changes in view or quality parameters change the resource consumption, and hence require readmission. There are several ways of reacting to a failed admission test. For instance, the user could be simply informed of the failure or the range of a quality slider could be restricted to an achievable value.

6.8   Prefetching for Interactive Use

Currently, prefetching is only used to ensure that the starting times of streams are synchronized with the rest of a presentation that is being played. The player reacts to user commands with a delay, which is the time it takes to request, to transmit and to decode the data needed. Maintaining prefetched data based on the current position in the presentation would allow immediate play start or display of the next frame when single-stepping. Providing a quick reaction to arbitrary changes of the position slider requires a sophisticated approach.

Chapter 7

Conclusions

This thesis has described the architecture of a multimedia player integrating support for advanced adaptive QoS control, complex presentations, and real-time transmission of continuous media streams from several remote servers.

Providing more flexibility in three QoS aspects of content, view, and quality opens new ways of watching multimedia presentations beyond the TV-like features we are used to. Authors are no longer confined to one video image and may present more information, allowing the viewer to observe the combination of different streams as well as simply to choose the most interesting one. Support for presentations with distributed components eliminates the need to download multimedia content before viewing enabling quick access for authoring and retrieval. Customizable view parameters allow users to adapt presentations to their needs. Lastly, multi-dimension quality control based on an error interpretation helps using resources efficiently. Hence, effective presentations can be achieved at low cost. The examples in Section 1.2 have shown that there is a need for these features. There is a potential for many other uses once users get familiar with the new possibilities.

7.1 QoS Model

The design of this player is based on the model of independent content, view, and quality notions. While the model has been used before to control resource requirements of a local player, this research has shown that it is also applicable and useful for controlling and structuring distributed and adaptive multimedia applications. Keeping content, view , and quality orthogonal in accordance with the model, however, may require some addi-

tional implementation effort and cause run-time overhead. To achieve independence of view and quality in the spatial dimension, for instance, an additional scaling step was required. Moreover, while resource consumption ideally corresponds to the quality parameters and is independent of the view, this assumption does not hold in general. While the assumption is true in our player in the temporal dimension. it is false in the spatial dimension: Frame rate affects resource requirements while play speed does not, whereas resolution and image size affect CPU consumption to a similar extent.

7.2   Control over Multiple Quality Dimensions

Control over several quality dimensions can improve resource utilization, that is provide more quality per resource. In the current implementation, temporal as well as spatial resolution are variable. Because resource consumption can be controlled by both parameters, there are several ways of adapting to the current resource availability. Choosing the one best for the user maximizes achievable quality. The adaptation currently works semi-automatically: the user manually controls resolution while a feedback mechanism adjusts the frame rate to the amount of resources available.

This infrastructure can be used to integrate both dimensions in an automatic adaptation technique based on an error interpretation for assessing the possible QoS configurations. To do so, the application needs to know what quality configurations are achievable. Hence, it needs to be able to reliably estimate the resource requirements of every configuration. The experiments with two quality dimensions (frame rate and resolution) and two resources (bandwidth and client CPU) described in Chapter 5, particularly Figure 19 and 20, have shown that changes of quality parameters affect each resource in a different way, with the respective bottleneck resource determining what quality changes are possible. That changing a quality parameter can shift the bottleneck makes it even harder to model resource consumption accurately.

7.3   Complex Distributed Content

The development of this player has shown that presenting complex distributed content is possible. Two important operators for composing presentations are concatenation of sequential streams and synchronization of concurrent streams. In the current implementation, these operators can be applied to streams from different remote servers, allowing the content of a presentation to be distributed. It is necessary to ensure synchronization of several streams that are being played at the same time as well as synchronization of start-up and end of streams with the rest of a presentation. While the particular mechanisms needed for supporting complex content largely depend on the underlying single-stream architecture, it is very likely for the resulting player to be more than a simple extension of the single-stream case. In developing our system several architecture-specific problems were encountered.

In our player, the server sends frames at a certain rate after receiving a play command. Once the transmission has started, a feedback mechanism ensures client-server synchronization. Scheduling the start of a stream, however, is hard. Unpredictable latencies in communication with the server do not allow scheduling the sending of the command some time ahead in a way that would make sure that the data arrive on time. In practice it works well to start the server some constant time ahead and prefetch data, but this approach does not work always. The latency still may be larger than the constant time used.

For playing several streams (with potentially different characteristics such as frame rate or resolution) in parallel, it is necessary to carefully separate per-stream structures from stream-independent structures, with structures including variables, buffers, windows, and processes. In our architecture, an entire video pipeline is replicated for every stream. The streams share a control part interacting with the user and providing a common timer.

References

[1]   R. T. Apteker, J. A. Fisher, V. S. Kisimov, and H. Neishlos. Video Acceptability and Frame Rate. *IEEE Multimedia*, Vol. 2, No. 3, Fall 1995, pp.32-40.

[2]   D. P. Anderson. Meta-scheduling for distributed continuous media. Technical report CB/CSD 90/599, University of California, Berkeley, October 1990.

[3]   D. P. Anderson, S.-Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews. Support for continuous media in the DASH system. In *Proc. of the 10th International Conference on Distributed Computing Systems* (ICDCS10), Paris, May 1990, pp. 54-61.

[4]   S. Cen, C. Pu, R. Staehli, and J. Walpole. A distributed real-time MPEG video audio player. In Proc. of the 5th International Workshop on *Network and Operating Systems Support for Digital Audio and Video*, Durham, New Hampshire, April 1995. LNCS 1018, Springer Verlag, Berlin, pp. 142-153.

[5]   Z. Chen, S.-M. Tan, R. H. Campbell, Y. Li. Real time video and audio in the world wide web. In *World Wide Web Journal*, Vol. 1, January 1996.

[6]   S. T.-C. Chou and H. Tokuda. System support for dynamic QoS control of continuous media communication. In Proc. of the 3rd International Workshop on *Network and Operating System Support for Digital Audio and Video*, La Jolla, California, November 1992. LNCS 712, Springer Verlag, Berlin, pp. 363-368.

[7]   C. L. Compton and D. L. Tennenhouse. Collaborative load shedding for media-based applications. In Proc. of the *International Conference on Multimedia Computing and Systems*, Boston, Massachusetts, May 1994, pp. 496-501.

[8]   L. Delgrossi and L. Berger, Editors. Internet Stream Protocol version 2 (ST2), protocol specification - version ST2+. Internet request for comments: 1819, RFC-1819, August 1995.

[9]   L. Delgrossi, C. Halstrick, D. Hehmann, R. G. Herrtwich, O. Krone, J. Sandvoss, and C. Vogt. Media scaling for audiovisual communication with the Heidelberg Transport System. In *Proc. ACM Multimedia 93*, ACM Press, New York, 1993, pp. 99-104.

[10]   G. D. Drapeau. Synchronization in the MAEstro multimedia authoring environment. In *Proc. ACM Multimedia 93*, ACM Press, New York, 1993, pp. 331-339.

[11]   D. Ferrari, J. Ramaekers, and G. Ventre. Client-network interactions in quality of service communication environments. In Proc. of the 4th *IFIP Conference on High Performance Networking*, Laige, Belgium, Dec. 1992, pp. E1-1 - E1-14.

[12]   R. Govindan and D. P. Anderson. Scheduling and IPC mechanisms for continuous media. In Proc. of the *Thirteenth ACM Symposium on Operating Systems Principles*, Pacific Grove, CA, October 1991, pp. 68-80.

[13]   L. Hardman, G. van Rossum, and D. C. A. Bulterman. Structured multimedia authoring. In *Proc. ACM Multimedia 93*, ACM Press, New York, 1993, pp. 283-289.

[14]   D. Hehmann, R. G. Herrtwich, W. Schulz, Th. Schuett, and R. Steinmetz. Implementing HeiTS: Architecture and implementation strategy of the Heidelberg High-Speed Transport System. In Proc. of the 2nd International Workshop on *Network and Operating System Support for Digital Audio and Video*, Heidelberg, Germany, November 1991. LNCS 614, Springer Verlag, Berlin, pp. 33-44.

[15]   R. G. Herrtwich. The role of performance, scheduling, and resource reservation in multimedia systems. In *Operating Systems of the 90s and Beyond*, A. Karshmer and J. Nehmer, editors, LNCS 563, Springer Verlag, Berlin, 1991, pp. 279-284.

[16]   M. E. Hodges, R. M. Sasnett, and M. S. Ackerman. A construction set for multimedia applications. *IEEE Software*, Vol. 6, No. 1, January 1989, pp 37-43.

[17]   D. Hutchison, G. Coulson, A. Campbell, and G. S. Blair. Quality of Service management in distributed systems. In *Network and Distributed Systems*, 1994, Addison-Wesley, editor M. Sloman, pp. 213-302.

[18]   K. Jeffay, D. L. Stone, T. Talley, and F. D. Smith. Adaptive, best-effort delivery of digital audio and video across packet-switched networks. In Proc. of the 3rd International Workshop on *Network and Operating System Support for Digital Audio and Video*, La Jolla, California, November 1992. LNCS 712, Springer Verlag, Berlin, pp. 3-12.

[19]   D. Le Gall. A video compression standard for multimedia applications. *Comm. ACM*, Vol. 34, No. 4, April 1991, pp. 46-58.

[20]   Th. D. C. Little and A. Ghafoor. Synchronization and storage models for multimedia objects. *IEEE Journal on Selected Areas in Communication*, Vol 8., No. 3, April 1990, pp. 413-427.

[21]   D. Maier and J. Walpole. Personal Communication

[22]   D. Maier, J. Walpole, and R. Staehli. Storage System Architectures for Continuous Media Data. In proceedings of *Foundations of Data Organization and Algorithms, FODO '93*, LNCS 730, 1993, Springer-Verlag, pp. 1-18.

[23]   H. Massalin and C. Pu. Fine-grain adaptive scheduling using feedback. *Computing Systems*, Vol. 3, No. 1,Winter 1990, pp. 139-173.

[24]   Th. Meyer-Boudnik and W. Effelsberg. MHEG explained. *IEEE Multimedia*, Vol. 2, No. 1, Spring 1995, pp. 26-38.

[25]   K. Nahrstedt and J. M. Smith. The QOS Broker. *IEEE Multimedia*, Vol. 2, No. 1, Spring 1995, pp. 53-67.

[26]   K. Patel, B. C. Smith, and L. A. Rowe. Performance of a software MPEG video decoder. In Proc. of *ACM Multimedia 93*, Anaheim, California, August 1993, pp. 75-82.

[27]   C. Pu and R. M. Fuhrer. Feedback-based scheduling: a toolbox approach. Proc. of the *Fourth Workshop on Workstation Operating Systems*, Napa Valley, California, October 1993, pp.124-128.

[28]   L. A. Rowe, K. D. Patel, B. C. Smith, and K. Liu. MPEG video in software: representation, transmission, playback. In *High Speed Networking and Multimedia Computing*, IS&T/SPIE, Symp. on Elec. Imaging Sci. & Tech., San Jose, California, February 1994, pp. 134-144.

[29] L. A. Rowe and B. C. Smith. A continuous media player. In Proc. of the 3rd International Workshop on *Network and Operating System Support for Digital Audio and Video*, La Jolla, California, November 1992. LNCS 712, Springer Verlag, Berlin, pp. 376-386.

[30] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. Internet Engineering Task Force, Audio-Video Transport Working Group, RFC-1889, Jan. 1996.

[31] R. Staehli. *Quality of Service Specification for Resource Management in Multimedia Systems*. Ph.D. thesis, Oregon Graduate Institute of Science & Technology, January 1996.

[32] R. Staehli and J. Walpole. Constrained Latency Storage Access. *Computer*, March 1993, pp. 44-53.

[33] R. Staehli, J. Walpole, and D. Maier. Quality of Service Specification for Multimedia Presentations. *Multimedia Systems*, November 1995, Vol. 3, No. 5/6, pp. 251-263.

[34] R. Steinmetz. Analyzing the Multimedia Operating System. *IEEE Multimedia*, Vol. 2, No. 1, Spring 1995, pp. 68-84.

[35] R. Steinmetz and C. Engler. Human perception of media synchronization. Tech. Report 43.9310, IBM European Networking Center, Heidelberg, 1993.

[36] T. Talley and K. Jeffay. Two-dimensional scaling techniques for adaptive, rate-based transmission control of live audio and video streams. In Proc. of *the Second ACM International Conference on Multimedia*, San Francisco, Oct. 1994, ACM Press, pp. 247-254.

[37] H. Thimm and W. Klas. δ-Sets for optimized reactive adaptive playout management in distributed multimedia database systems. In Proc. of the *12th International Conference on Data Engineering*, New Orleans, Louisiana, Feb. 1996, pp. 584-592.

[38] H. Tokuda, Y. Tobe, S. T.-C. Chou, and J. M. F. Moura. Continuous media communication with dynamic QOS control using ARTS with an FDDI network. ACM SIGCOMM 92, Baltimore. *Computer Communications Review*, Vol. 22, No. 4, October 92, pp. 88-98.

[39] A. Vogel, B. Keherve, G. v. Bochmann, and J. Gescei. Distributed multimedia and QOS: A survey. *IEEE Multimedia,* Vol. 2, No. 2, Summer 1995, pp.10-19.

[40] R. Weiss, A. Duda, and D. K. Gifford. Composition and search with a video algebra. *IEEE Multimedia,* Vol. 2, No. 1, Spring 1995, pp. 12-25.

[41] B. Wolfinger and M. Moran. A continous media data transport service and protocol for real-time communication in high speed networks. In Proc. of the 2nd International Workshop on *Network and Operating System Support for Digital Audio and Video,* Heidelberg, Germany, November 1991. LNCS 614, Springer Verlag, Berlin, pp. 171-182.

[42] L. Zhang S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource ReSerVation Protocol. *IEEE Network,* Vol. 7, No. 5, Sept. 1993, pp. 8-18.

Biographical Note

Rainer Koster was born in Trier, Germany, on September 19, 1970. He attended the Friedrich Wilhelm Gymnasium in Trier and received the Abitur in 1990. From 1991 to 1995 he studied at the University of Kaiserslautern, Germany. He was awarded a scholarship of the Cusanuswerk, Bonn, Germany in 1994. He was a teaching assistant in Numerical Mathematics, and a research assistant in the Systems Software Group. In Fall 1995 he joined the Master's program in Computer Science at the Oregon Graduate Institute choosing the thesis option in the winter quarter 1996. His research interests include distributed operating systems and multimedia systems.