Iterative Transmission of Media Streams

Michael Merz, Konrad Froitzheim, Peter Schulthess, Heiner Wolf
University of Ulm, Germany
{merz, frz, schulthe, wolf}@informatik.uni-ulm.de

Abstract

Multimedia streams are difficult to transmit in full quality and real-time due to their size and constraints imposed by their timely presentation. Especially in slow channels with variable throughput - such as typical Internet connections - media streaming is extremely hard. We introduce a new iterative method for the transmission of media streams. The idea is to match each iteration of the transmission process exactly to the play-out time of the respective media stream, thus allowing synchronous presentation of the stream's contents during the transmission. The quality of the playback improves with the number of completed iterations, from rough during the first pass to full quality at the end. The WebMovie system implements this iterative streaming scheme for video data. After the introduction of the system, performance measurements are discussed and compared to conventional methods of movie transmission.

Keywords

Iterative transmission; real time transmission; full quality transmission; layered media encoding; Internet; WWW; multimedia; media streams; scaleable media streams; refineable media streams; data segmentation; WebMovie.

1 Introduction

The World Wide Web (WWW) has evolved from a static document-oriented Internet service to a feature-rich multimedia system. Audio and video have become part of many Web pages; however, their integration has not been completed yet: many issues in the area of timely delivery to the client remain unsolved.

1.1 The File Transmission Paradigm

The main problem for the transmission of multimedia streams via the Web is the severe limitation in bandwidth. Although the available data rate may be exploited efficiently with sophisticated compression techniques, even short media clips with a playback duration of seconds (e.g. short audio- or video-clips) are typically of considerable size ($\approx 10^6$ Bytes). Hence, their transmission time is typically much longer than the presentation of the data to the user. Consequently, it is impossible to present the data in real-time during a transfer according to the commonly used *file transmission paradigm*. Typical systems transfer the clip before they start the playback.

1.2 Scaleable Media Streams

In order to make real-time transmissions of multimedia data possible despite its volume and despite low-bandwidth connections, recent software systems like WebAudio/WebVideo (see [19]), WaveVideo (see [4]) and many others avoid transmitting media streams (i.e. video) in full quality. Instead, they adjust the media quality - and with it the amount of the transferred data - to the available bandwidth. In other words, the lower the bandwidth, the poorer the quality of the transmitted data's playback in terms of temporal and/or spatial resolution. Furthermore, error correction is not performed - retransmission due to packet losses would impair the presentation.

Media streams suitable for these schemes are called *scaleable media streams* because the amount of data to be transmitted is scaled according to the currently available bandwidth. The recent MPEG-2 standard (see [8]) supports the transmission of scaleable video streams by a layered encoding of the data. Scaleable media streams are considered well-suited for many applications demanding live transmission of video and audio streams. For other applications with clients requesting media data in full quality (e.g. video on demand), scaleable media streams are not sufficient due to the inevitable scaling-loss.

1.3 Refineable Media Streams

In a sense, our approach is an extension of scaleable media streams: based on the idea of adjusting the stream to the available throughput, the streams are filtered to achieve *real-time* presentation during the transmission. However, the data filtered out is not discarded, but transferred later to ensure complete delivery (see listing 1).

Transmission in *full quality* even through low-bandwidth connections is achieved by folding the transfer time into the presentation time: if a prerecorded media stream is too large to be streamed in real-time due to limited bandwidth, it is broken up into several layers. The layers have similar sizes, but differ in their significance for the presentation. Each of them represents a partial stream and needs less bandwidth for timely transmission than the original stream (see figure 1).

ACM Multimedia *97 Seattle Washington USA*

```
...
Open (dataChannel, UDP);
Open (cntlChannel, TCP);
ResetAcks ();
while (∃ unsent frame) do
   nextFrame := SelectNextFrame (bandwidth);
   Transmit (dataChannel, nextFrame);
   if (Receive (cntlChannel, ack, frame))
      UpdateAcks (frame);
      bandwidth := EstimateBandwidth ();
   fi
od
Close ();
...
```

listing 1: code fragment of a server for iterative transmission of movies (using vertical segmentation only; see chapter 3);

In the first iteration we deliver as many layers as possible according to the available bandwidth - similar to scaleable media streams (see [4], [5]). But in contrast to scaleable media streams, the transmission does not stop here: in subsequent iterations, the layers neglected in the first iteration are delivered.

The most significant layer with the basic information is transmitted first, followed by the others with descending significance for the presentation. The first layer's data allows the presentation of the original stream's contents - even though in rather limited quality. The subsequent layers' information refines the basic information, improving the quality of the playback step by step. Finally, the arrival of the last layer completes the data for the client and allows a presentation and even subsequent processing in full quality. Hence, a stream that allows an iterative transmission is called a *refineable media stream*; an exact definition is given in section 2.1.

Although the subsequently introduced methods for iterative streaming were initially developed to design a transmission technique for media data via low-bandwidth channels, it has turned out that they are quite useful in other fields of application, too: if not the complete data of a stream is required, but only a characteristic fingerprint is needed, storage capacity and time may be saved by storing only a limited number of layers instead of the complete stream. Moreover, the reduced data volume of these fingerprints, e.g. in (movie-) databases, may improve search times and therefore the access speed.

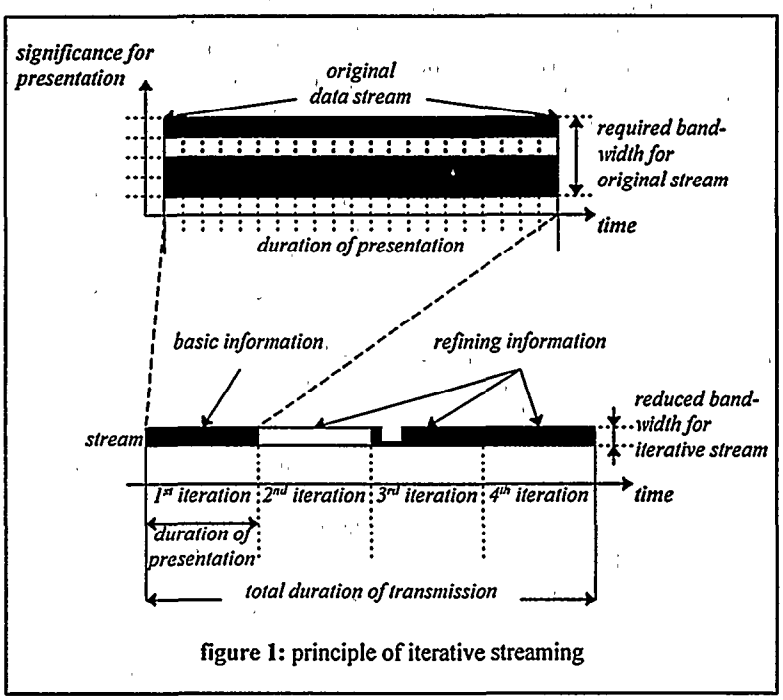

figure 1: principle of iterative streaming

1.4 Overview

The rest of the paper is organized in four groups: section 2 introduces new techniques for iterative streaming of time critical media data. Chapter 3 describes the recently developed WebMovie system to illustrate the streaming of video data using vertical segmentation and presents the experience with the implementation and operation of this system. Subsequently, we present measurements of the performance of the WebMovie system. Section 4 first summarizes the most important issues presented and then draws conclusions. Further work is then outlined in chapter 5.

2 Refining Media Streams

The algorithms introduced in this paper may be applied to almost all prerecorded real-time media streams. The most common cases are probably video and audio streams. Again: the idea is not to transmit the respective stream byte by byte, but folded in time. Since we do deliver prerecorded data, it is possible to transmit all data excluded in the first iteration of the transmission in further phases. The stream is presented several times during the transmission in steadily improving quality: the stream is refined iteratively. This allows the users to follow as many iterations of the presentation as they like to. Users may interrupt the transmission at any time - even if the transmission has not been completed. Towards the completion of the transmission, during the last iteration, the presentation reaches full quality. Transmitting media streams this way creates the idea of the play-out time of the streams' contents *unfolding* into the transmission time (see figure 1).

After a short definition of refineable media streams several approaches to the iterative transmission of these streams are presented.

2.1 Definition

A media stream is called *refineable* if and only if it fulfills the following conditions:

- *All stream data is well-known.* In other words, the data to be transmitted must not be discarded right after the transmission of the first iteration.
  This condition is fulfilled in the case of prerecorded video and audio data (i.e. movies). Live transmissions do not satisfy this constraint because data is created dynamically. Hence, the stream is not finite and the end of the stream is unknown. However, all data might be recorded to refine the stream in

later iterations in order to apply iterative streaming to live streams, too.

- *Information atoms* are postulated, representing the smallest units of information in the respective stream.[1] These information atoms either may be given by the nature of the respective stream, e.g. the single frames of a video stream, or they may be artificially defined, e.g. groups of sound samples in an audio stream.
  This condition is easy to be fulfilled by media streams because (common) computers cannot work with continuous signals: typically, signals are digitized before being processed.

2.2 Vertical Segmentation

Probably the easiest way to split the information of a refineable media stream into several layers lies in the time domain: a subsampling of the respective stream yields the first layer.[2] The resulting stream resembles a scaleable media stream. All further layers are built in the same way as the first layer. Each layer contains only samples the other layers do not contain, of course (see figure 2). For the sake of an efficient error correction, exceptions from this rule may be made (see section 3.4). This subsampling method is referred to as *vertical segmentation.*

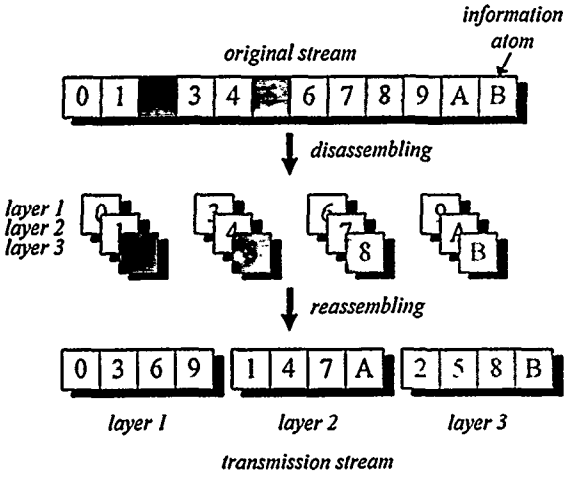

figure 2: vertical segmentation of a refineable generic media stream

There are two ways to disassemble a media stream into layers according to the vertical segmentation:

- The whole layer generation process may be performed *statically* in advance of the transmission. Number and size of the single layers have to be fixed before the beginning of the transmission and cannot be adjusted afterwards.
  For well-known transmission channels, heuristics may be employed in order to determine the size of the single layers. A minimal quality-of-service might be presumed, for instance.
- The alternative to generating the layers statically is their *dynamic* calculation: The available bandwidth is continually measured and the subsampling rate is adapted to it. In other words, each layer is composed dynamically, on-the-fly, while it is being transmitted (see [5]). The single information atoms are delivered *just-in-time.*

With the static approach there is no way to adapt the transmission parameters (i.e. the subsampling rate) to the actually available bandwidth during the transmission. If the bandwidth significantly decreases - e.g. due to common fluctuations of bandwidth somewhere in the Internet - concurrent real-time playback and transmission may not be achieved anymore. On the other hand, if the bandwidth increases above the estimate, it cannot be exploited and will be wasted.

The more sophisticated and flexible approach to vertical segmentation, the dynamic scheme, offers significantly more potential. Due to the continual observation of several network parameters (e.g. bandwidth), the dynamic variant is more complicated. Depending on the type of media and the implementation, some problems may arise as discussed for video streams in chapter 3.

Using vertical segmentation, the transmission of a media stream may be performed concurrently with the presentation of its contents, which comes at the price of jerky presentation during the first iterations. After the completion of the transmission, the stream's playback corresponds to the original.

2.3 Horizontal Segmentation

Another method that may be used in order to disassemble refineable media streams is to break up the information atoms themselves. As postulated in section 2.1, an atom itself will not be broken up any further, but its representation may. In other words, the total amount of information stored in an atom is not sent in one single step, but is distributed over several layers. Virtually, what is refined are not the media streams themselves, but the single information atoms. First, a rough approximation of each atom is transmitted and then refined in subsequent iterations of the transmission. This procedure can be applied to each atom and is referred to as *horizontal segmentation* (see figure 3).

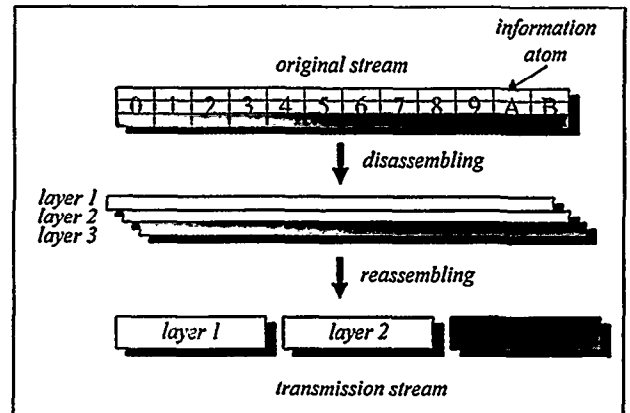

figure 3: horizontal segmentation of a refineable generic media stream

Compared to the vertical segmentation, the advantage of this method is improved scaleability: not only may whole atoms be omitted in a layer, but the size of the atoms themselves may be scaled. This allows an almost perfect adaptation of the data rate to

[1] However, the internal representation of these atoms can possibly be broken up further (see section 2.3).

[2] This subsampling is always possible due to the postulation of information atoms (see section 2.1). It may be performed at least based on the respective stream's atoms.

the available bandwidth. However, with most easy-to-implement approaches to horizontal segmentation (e.g. defining the color components of video frames as layers), the limits of adapting the transmission to the bandwidth will be reached rather soon. Hence, the information atoms should be defined carefully.

Again, iterative transmission of a media stream and its concurrent presentation can be achieved with horizontal segmentation. Since the resolution of the single information atoms is rather low in the beginning of the transmission, the playback of the stream's contents will be rather poor during the first iterations - depending on the available bandwidth. Consequently, the first replays may look as if a low-pass filter had been applied to the atoms. As with the vertical segmentation, the quality of the presentation rapidly improves - up to that of the original stream after the transmission has been completed.

2.4 Hybrid Segmentation

A third approach to segmentation, the *hybrid segmentation*, combines the two techniques presented above and is the most flexible. Again, there are two possibilities to perform the merging:

- The approach easier to implement is to apply the vertical segmentation to the original stream first. Then the horizontal segmentation is applied to the layers generated in the first step, yielding sublayers of each layer (see figure 4).[3] This method is referred to as *VF[4]-hybrid segmentation.*
- The alternative is to apply the horizontal segmentation first and then the vertical segmentation to the low-quality atoms afterwards (*HF[5]-hybrid segmentation*, see figure 5).[6]

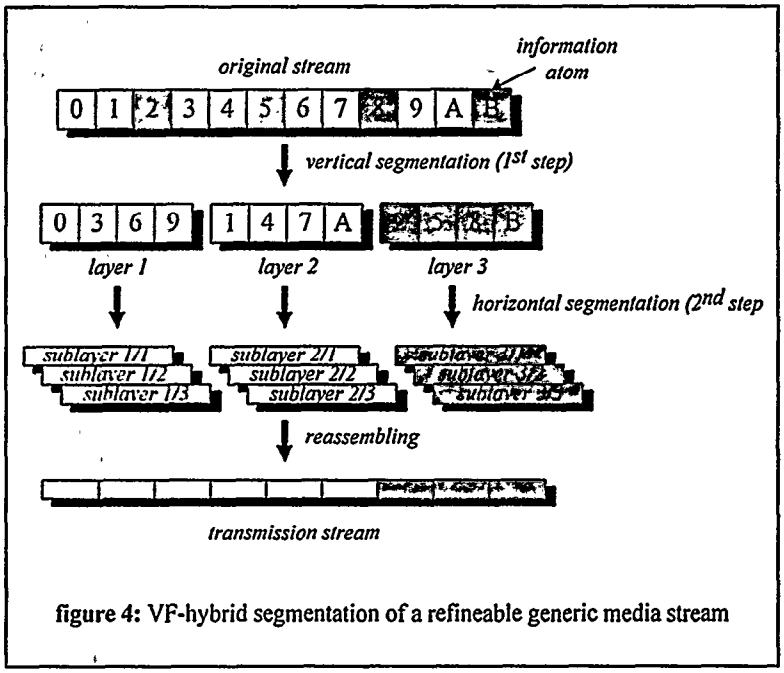

figure 4: VF-hybrid segmentation of a refineable generic media stream

Combining both, vertical and horizontal segmentation, the hybrid segmentation provides a flexible and efficient strategy for disassembling media streams. Allaying the drawbacks of both techniques the advantages of both of them may be exploited: the effect is minimizing the gaps between the single information atoms transmitted in the first iteration (due to the subsampling of the vertical segmentation), and maximizing the initial quality (i.e. resolution) of these atoms (which would be rather poor if only horizontal segmentation was used). The segmentation is said to have a finer *granularity*, which allows reacting carefully to fluctuations of the available bandwidth. Furthermore, the hybrid segmentation allows real-time transmission of media streams in full quality via connections with very low bandwidths, which would otherwise cause vertical and horizontal segmentation alone to fail.

These advantages come at the price of significant overhead and a complex implementation: sophisticated book-keeping is needed to track which data has been sent and which has been acknowledged already. If the implementation uses an unreliable transport protocol like UDP - as strongly recommended to avoid retransmission delays - and performs the necessary error correction on its own,[7] even the error correction gets complicated.

2.5 Summary

The hybrid segmentation is the most sophisticated, flexible, and elegant way to perform the disassembly of media streams. On the other hand the implementation, especially regarding the necessary error correction, is not trivial at all. The retransmission overhead grows dramatically and the algorithm becomes inefficient. Moreover, it has turned out that for practical use such as the transmission of common video data (e.g. using the Common Interchange Format) through low-bandwidth channels (with current V.34 modems (33.6 kbps)), the employment of either vertical or horizontal segmentation suffices.

Recently, considerable effort has been invested in the field of real-time transmission of bulky media streams via low-bandwidth networks - namely the Internet. Basically, they use techniques such as bandwidth negotiation (with and without renegotiating), or layered respective scaleable media streams (see [4], [10]). All of them sacrifice full-quality delivery of the data if the available bandwidth drops below a certain limit. Instead, as much data is delivered as the available bandwidth allows for. This comes at the price of a reduced quality of the presentation.

We introduced a new class of algorithms for transmitting time critical media data in both full quality and real-time via low bandwidth connections. Three subclasses of these methods were outlined.

[3] An already existing vertical segmentation may be subsequently extended very easily by horizontal methods.

[4] vertical first

[5] horizontal first

[6] It has turned out that this procedure requires much more bookkeeping than the VF-hybrid, since the information atoms have been broken up in the first step already.

[7] Since the complete stream information needs to be transmitted without omitting a single byte, data retransmission mechanisms must be provided - in contrast to scaleable media streams.

3 The WebMovie System

Having introduced refineable media streams in chapter 2, we will give an overview over the WebMovie system (see [12], [13]) in this chapter. WebMovie implements refineable media streams with vertical segmentation. It focuses on the full-quality delivery of video data. The information atoms (see section 2.1) are defined as the single frames within a video stream.

3.1 The Architecture

The WebMovie system consists of three components:

- the *preprocessor* prepares the movie data prior to the transmission. The preprocessor computes relevant information offline and stores it in a database for future use.
- the *server* transmits a video stream's contents in consecutive iterations (see figure 1).
- the *client* receives the successive iterations of the transmission, recombines them into one video stream (i.e. M-JPEG) and presents the movie in real-time concurrently with the reception.

The server and the client communicate via two communication channels: the *data channel* is used for the transmission of the movie data, the *control channel* for the transmission of acknowledgement and control information. The control channel uses the TCP protocol, which ensures the correct in-order delivery of all data transmitted. This is very important for the sensitive control data. The data channel uses unreliable UDP in order to implement a sophisticated error-correction scheme in the WebMovie system (see section 3.4). Moreover, the data rate may be chosen by the server.

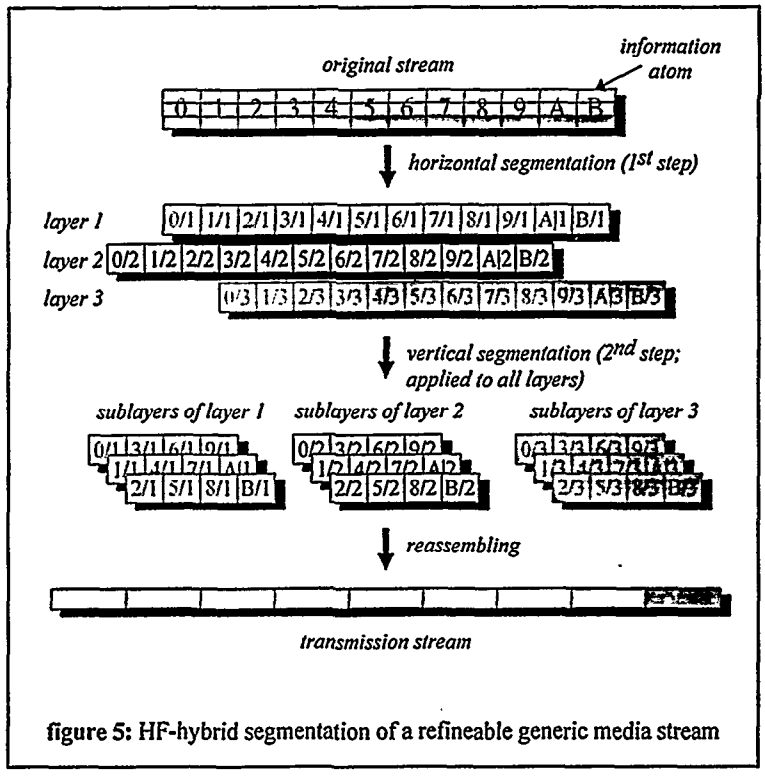

figure 5: HF-hybrid segmentation of a refineable generic media stream

3.2 Data Compression

Several data compression standards are available to encode the image data. The most important ones are methods for compressing single frames like GIF (see [3]), JPEG (see [7]) on the one hand, and moving picture compression techniques like H.261 (see [9]), or MPEG (see [6]) on the other hand.

To transmit time-critical media streams, the reduction of the data volume in advance of its transmission is considered the most important issue. Standards like H.261 and particularly MPEG seem to present the best solution: eliminating the temporal redundancy between the single frames of a video stream almost completely, compression ratios from 50:1 (H.261, see [16]) up to 200:1 (MPEG, see [16]) may be achieved. The shortcomings of these compression algorithms are that they produce strong dependencies between the single frames. In other words, decompressing one (differentially coded) frame typically requests the decompression of other (intra-coded) frames, too. The better the compression ratio, the stronger these dependencies. Real random access to the single frames of a video stream is not possible.[8]

We decided to use JPEG - a compression standard for single frames. The compression ratio may reach from 8:1 (almost original quality) up to 50:1 (poor quality). JPEG does not produce any dependencies between the single frames, allowing the transmission of the frames in a random order - as needed for vertical segmentation.

In order not to forego the drastically improved compression ratio of temporal redundancy reduction, we use an action block algorithm in combination with JPEG[9]: each of the 8x8-blocks of a JPEG-coded frame is compared to the corresponding block of the frame next to this frame, which has been acknowledged already by the client. If no significant changes can be detected, the respective block will not be transmitted, but reused from the already transmitted frame. This additional method allows a significant reduction of the data volume, although we do not achieve compression ratios in the range of MPEG. MPEG uses motion vectors in addition to differential encoding.

The big difference to H.261 and MPEG is the on-the-fly removal of the temporal redundancy. Moreover, the reference frame for each frame to be encoded differentially may be chosen on-the-fly considering the frames that have already been transmitted and the available bandwidth. The latter is an important difference to the schemes found in the literature: our compression algorithm refers to the last frame transmitted, which is not necessarily the predecessor in the movie.

[8] We considered several publications (e.g. [14], [15], [17]) on software providing full VCR functionality for MPEG-encoded data. All of them have to decode several frames in order to get a B-frame, for instance because the information atoms of an MPEG stream are the „groups-of-pictures" (see [6]), which, however, are too large for our purposes.

[9] Our action block approach is very similar to the „Intra-H.261" as proposed in [11].

3.3 Dynamic Vertical Segmentation

It has turned out that some problems arise if vertical segmentation is used in combination with a dynamic layer generation: With the dynamic layer generation, the frame to be transmitted next has to be calculated carefully in order to make its just-in-time delivery possible (as required for real-time transmissions). This calculation crucially depends on the estimated bandwidth and on the estimated size of the frame to be transmitted next:

The bandwidth is estimated with probe-echo based methods measuring the round-trip-delay as proposed in many publications (see e.g. [18]).

Estimating the size of the frame to be delivered next is significantly more complicated: the size of a frame depends on the reference frame used for its differential encoding (see section 3.2). It is necessary to encode all frames that are candidates for being sent next - just to get their sizes. Moreover, this information is not useful anymore as soon as the chosen frame has been sent because the frame sizes change if they are encoded differentially using another frame as reference. Hence, the single frames cannot really be delivered exactly just-in-time, only approximately. Obviously, the approach to encode all candidates for transmission differentially is highly inefficient. Our straight-forward approach is to estimate the frame size based on the sizes of its predecessors; this linear predictor has proven to be an adequate heuristic to estimate a differential frame's size.

This method does not work well with scene changes in movies. The reason is that the difference between a picture before a cut and a picture afterwards is typically drastic. Nearly the entire frame will change and almost no block will be the same. There are two ways to handle the cut-problem:

- It can simply be ignored. This is the way the current prototypical WebMovie system treats the problem. WebMovie just softens the real-time condition for the presentation resulting in a snap-to-grid approach: if the frame that is to be presented next according to the real-time paradigm has not arrived yet, the closest predecessor in the movie is presented instead. Hence, frames that arrive too late for their immediate presentation according to the common real-time paradigm may be presented anyway. This allows a rapid improvement of the presentation's quality - especially for long movies - despite the fact that there is no way to deliver frames really just-in-time.
- The alternative and strongly recommended way is to employ an efficient cut-detection algorithm, as e.g. introduced in [2]. The cuts might be detected in the preprocessing step, similar to the calculation of the data needed for the action-block method. The first iteration of the transmission might contain key frames, i.e. the first frames of the single scenes of the respective movie. In further iterations of the transmission, all other frames might then be encoded very efficiently according to the action-block method because the changes from one frame to another will typically be minor.

3.4 Delayed Data Retransmission

As mentioned above, using a transport protocol like TCP would foil all perspectives opened up by refineable media streams: the immediate, automatic error correction with data retransmission and mechanisms like „slow start", significantly slowing down the data rate after an error occurred, is not suitable for the transmission of time-critical media streams such as video streams. However, we have to employ data retransmission eventually because we want to deliver all data in full quality. The characteristics of iterative transmissions suggest special error-correction algorithms, such as the *delayed data retransmission*:

Packets lost due to the unreliable nature of UDP are not retransmitted immediately. Instead, we continue with transmitting the current iteration as if no data had been lost. Hence, it is possible to maintain the real-time presentation despite data losses. All data that has not been acknowledged for a certain span of time is considered „unsent". Since WebMovie composes the layers of the respective stream dynamically, lost data is automatically retransmitted in a later iteration of the transmission (see figure 6).

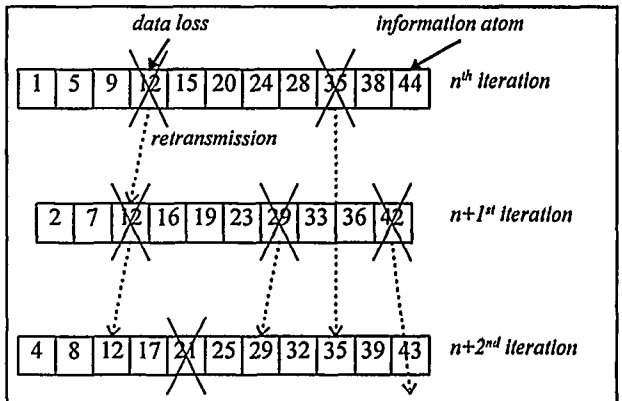

figure 6: principle of delayed data retransmission

After the transmission has been completed, all the data has been delivered. This elegant integration of error correction into the iterative movie transmission comes at no extra cost to the implementation. It actually reduces the algorithmic overhead compared to other schemes.

3.5 Experimental Results

3.5.1 The Testbed

Our testbed consists of the WebMovie server running on a PC with an Intel-Pentium-200 CPU and the client running on a PC with an Intel 80486-66 CPU. The connection is established using a V.34 modem, which allows data rates up to 33,600 bps. In our experiments we limited the bandwidth to 28,800 bps in order to get results valid for the majority of Internet users.

3.5.2 The Video-Clips

To measure the performance of our WebMovie system, we chose three different, short video-clips, all recorded from television:

- The *Al Bundy*-clip shows Al talking to Kelly. Then Bud and a girl are discovered in a closet. The camera often pans around in the room and the video contains several cuts. Therefore this frame-sequence is well suited to test the WebMovie system under worst-case conditions: from one frame to another almost all image-parts can change. There are hardly any stationary image-parts allowing the action-block algorithm to reduce the data volume to be transmitted.
- In the *Wallace & Gromit* cartoon, a penguin-burglar is stealing valuables when he is surprised by the owner. Since this clip consists of artificially generated scenes, parts of the

image are exactly constant - they do not change at all from one frame to the next. These rather small constant parts of the image have to be transmitted only once according to the action-block algorithm, so the data volume is reduced significantly. Several cuts in this clip keep the action-block method from working optimally.

- The third clip, a *Music Video*, shows an excerpt from the music video of Michael Jackson's song „Black or White". It mainly shows faces of „black and white" women and men morphing from one into another. The background, a blue wall, is (almost) constant. This clip was chosen because the changes between frames directly following each other are not too big. There is no cut in the whole movie. This is why the action-block algorithm can work properly - much better than in Al Bundy and better than in Wallace & Gromit, too.

The characteristics of the three movies are subsequently summarized in table 1.

table 1: characteristics of the test movies

	Al Bundy	Wallace & Gromit	Music Video
Num. of Frames	304	162	608
Frame Dims.	192x136	240x168	160x120
MPEG (KBytes)	1,057[10]	*unknown*[11]	766[10]
JPEG/M (KBytes)	1,632	931	1,427
Frame Rate (fps)	10	10	10

3.5.3 Performance Measurements

The parameters measured using the test movies introduced above are listed in table 2: the time from the start of the transmission until the presentation of the first frame and the time it takes to complete one whole iteration of the transmission (both measured in seconds). The next two lines contain the average number of frames transmitted during the first iteration and the total number of iterations of the transmission. Finally, the total transmission time for all movies was measured. This parameter is given relative to the time needed to transmit the MPEG-compressed movie sequentially with ftp. In the first approximation, the total transmission time is directly proportional to the amount of data actually transmitted. Since this parameter is given relatively only, it may be understood as the relative volume of the transmitted data compared to ftp/MPEG as well.

According to these results, the first frames of the movies are presented very quickly, only a few seconds after the beginning of the transmission. During the first iterations of the transmission, frames of all scenes of the respective movie are presented - *not* only frames of the movie's beginning. The transmission of every iteration of a movie takes exactly the same time as the movie's playback in real-time. Hence, the users may watch as many presentations in improving quality as iterations needed for the transmission. They may observe the progress of the transmission and abort it at any time.

[10] This movie has been converted with the *ImageMagick*-public domain software (available at ftp://sunsite.cnlab-switch.ch/mirror/MachTen/ applications/68k/X11/graphics) from the MPEG format into the motion JPEG format.

[11] This movie was initially motion JPEG coded.

These advantages concerning presentation-oriented transmitting of the movies are at the expense of the total transmission time, which increases about 30% compared to the transmission of the movies with standard methods (plain TCP, e.g. ftp) using the highly efficient MPEG-compression standard. Although not the focus of this research, the latter result is quite surprising: MPEG's advantage compared to our simple action block scheme is much smaller than anticipated.

table 2: results of the transmission of video streams with WebMovie (at 28.8 kbps)

	Al Bundy	Wallace & Gromit	Music Video
Presentation of 1st Frame	< 3 sec	< 5 sec	< 2 sec
Duration of 1st Iteration	30.4 sec	16.2 sec	60.8 sec
Number of Frames in 1st Iteration (average)	10	5	30
Number of Iterations	16	22	20
Transmission Time relative to MPEG/ftp	125%	< 400 sec[11]	135%

4 Summary and Conclusions

We have introduced the completely new iterative transmission paradigm, which is based on the ideas of scaleable media streams and combines the advantages of the two conventional paradigms: it makes real-time transmissions of media streams possible in combination with the delivery of the data in full quality - even via the Internet with its severely limited bandwidth.

We have shown several approaches to possible implementations of this new transmission paradigm and discussed the advantages and disadvantages of each of them.

Then we briefly introduced the WebMovie system as a showcase implementation of iterative streaming - focusing on the transmission of prerecorded video data. A new error-correction mechanism - the delayed retransmission - has been introduced, which supports the iterative nature of the employed transmission paradigm. Some measurements prove the potential of iterative streaming: only seconds after the beginning of the transmission first frames are presented - in real-time and not only frames from the beginning of the movie, but from all scenes of the movie. Due to an action-block algorithm, the data volume is reduced in comparison with the volume of the motion JPEG encoded movie.

It has also been shown that if the data is to be transmitted in full quality the transmission takes up about 130% of the time needed by using the conventional file transmission paradigm (e.g. FTP) - in combination with MPEG compression. However, the file transmission paradigm does not allow real-time presentation of the movie concurrently with the transmission, thus straining the patience of the user.

The client software - a Netscape plug-in, which is currently available for Windows95 only - can be downloaded at [13]. The reader is invited to test it with our WebMovie server (see [13]) in order to get a demonstration of iterative streaming.

Iterative streaming is well-suited for any field of application where the data is eventually needed in full quality, for instance for video on demand. Moreover, users may save a lot of time and

storage capacity: they may choose the quality in which they want to have the data. The transmission may be gracefully aborted at any point of time during the transmission. The data is not stored in full quality if the transmission is interrupted.

5 Future Work

The WebMovie system is currently implemented as a prototype only. Although all basic functionality is available in this prototype, there are many items to be improved and optimized. The following three paragraphs may give an idea of what needs to be done next to make the WebMovie system even more attractive:

- One of our future goals is to alleviate the main drawback of our compression scheme: the data volume to be transmitted is still higher than it could be using MPEG-compression. A further reduction of the data volume may be performed by adapting the encoding of the video data to be transmitted even more to the standards proposed in [6], [8], or [9], e.g. dynamically creating *motion vectors*.
- Another really interesting field of research is the *improvement of the scalability* of the iterative movie transmission. The current version of WebMovie only performs a segmentation of the movie information by sending the single frames out-of-order. Actually, there are many other methods to break up this information: instead of using the vertical decomposition of the movies' information into levels of frame rates, a horizontal decomposition into levels of image quality might be performed as well (see [12]). Improved JPEG-standard algorithms like successive approximation or spectral selection might be used (see [7]).
- The iterative strategy for data transmission shown in this paper has been applied to the WebMovie system, which has been designed to transmit video data via low bandwidth communication channels. As mentioned above, the strategy itself may be applied to almost *any type of media* - not only video.
  In order to provide a full movie transmission system, the current WebMovie system might be extended by facilities for the iterative transmission of high-quality audio streams. Audio may be delivered in telephony quality during the first iteration of the transmission - to be improved to high-fidelity in further iterations.

6 References

[1] Amir, E., McCanne, S., Zhang, H. „An Application-level Video Gateway". *Proceedings of the 3rd ACM International Conference on Multimedia.* San Francisco, CA, November 1995.

[2] Arman, F., Hsu, A., Chiu, M. „Image Processing on Compressed Data for Large Video Databases". *Proceedings of the 1st ACM International Conference on Multimedia.* August 1993.

[3] CompuServe Incorporated. *GIF89a Specification.* Columbus, OH, 1990. Software on-line.[12]

[4] Dasen, M., Fankhauser, G., Plattner, B. An Error Tolerant, Scalable Video Stream Encoding and Compression for Mobile Computing. *ACTS Mobile Summit 96.* Granada, Spain, November 1996.

[5] Hoffman, D., Speer, M., Fernando, G. „Network Support for Dynamically Scaled Multimedia Data Streams". *Proceedings of 4th International Workshop on Network and Operating System Support for Digital Audio and Video*: pp. 251-262. Lancaster, UK, 1993.

[6] ISO 11172. *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s.* International Organization for Standardization, Geneva, 1993.

[7] ISO 10918, ITU-T Rec. T.81. *Information Technology - Digital compression and coding of continuous-tone still images.* International Organization for Standardization, Geneva, 1992.

[8] ISO 13818. *Information technology - Generic coding of moving pictures and associated audio information.* International Organization for Standardization, Geneva, 1996.

[9] ITU-T Rec. H.261. *Video Codec for Audiovisual Services at px64 kbit/s.* International Telecommunication Union, Geneva, 1990.

[10] Krishnamurthy, A., Little, T.D.C. „Connection-Oriented Service Renegotiation for Scalable Video Delivery". *Proceedings of the 1st IEEE International Conference on Multimedia Computing and Systems*: pp. 502-507. Boston, MA, May 1994.

[11] McCanne, S., Jacobson, V. „vic: A Flexible Framework For Packet Video". *Proceedings of the 3rd ACM International Conference on Multimedia.* San Francisco, CA, November 1995.

[12] Merz, M. *Iterative Refinement of Video Streams.* MSc thesis, University of Ulm, Germany, 1996.

[13] Merz, M. *The WebMovie Homepage.* Software on-line.[13]

[14] Rowe, L. A., Smith, B. C. „A Continuous Media Player", *Proceedings of the 3rd International Workshop on Network and OS Support for Digital Audio and Video.* San Diego, CA, 1992.

[15] Rowe, L. A., Patel, K. D., Smith, B. C., Liu, K. „MPEG Video in Software: Representation, Transmission, and Playback". *Proceedings of the IS&T/SPIE High Speed Networking and Multimedia Computing.* San Jose, CA, February 1994.

[16] Smith, B. C. *A Survey of Compressed Domain Processing Techniques.* Software on-line.[14]

[17] Smith, B. C. *Implementation Techniques for Continuous Media Systems and Applications.* PhD thesis, University of California, Berkeley, CA, 1993.

[18] Tanenbaum, A. S. *Modern Operating Systems.* Prentice-Hall, 1992.

[19] Wolf, K. H., Froitzheim, K., Weber, M. „Interactive Video and Remote Control via the World Wide Web". *Interactive Distributed Multimedia Systems and Services*: pp. 91-104. Springer, Berlin, 1996.

[12] http://www.w3.org/pub/WWW/Graphics/GIF/spec-gif89a.txt

[13] http://www-vs.informatik.uni-ulm.de/Mitarbeiter/Merz/webmovie/index.html

[14] http://www.uky.edu/~kiernan/DL/bsmith.html