Multimedia Systems (1994) 2:172–180

Multimedia Systems

Media scaling in a multimedia communication system

Luca Delgrossi, Christian Halstrick, Dietmar Hehmann, Ralf Guido Herrtwich, Oliver Krone, Jochen Sandvoss, Carsten Vogt

IBM European Networking Center, Distributed Multimedia Solutions, Vangerowstrasse 18, D-69115 Heidelberg, Germany

Abstract. HeiTS, the Heidelberg Transport System, is a multimedia communication system for real-time delivery of digital audio and video. HeiTS operates on top of guaranteed-performance networks that apply resource reservation techniques. To make HeiTS also work with networks for which no reservation scheme can be realized (for example, Ethernet or existing internetworks), we implement an extension to HeiTS which performs media scaling at the transport level: The media encoding is modified according to the bandwidth available in the underlying networks. Both transparent and nontransparent scaling methods are examined. HeiTS lends itself to implement transparent temporal and spatial scaling of media streams. At the HeiTS interface, functions are provided which report information on the available resource bandwidth to the application so that nontransparent scaling methods may be used, too. Both a continuous and discrete scaling solution for HeiTS are presented. The continuous solution uses feedback messages to adjust the data flow. The discrete solution also exploits the multipoint network connection mechanism of HeiTS. Whereas the first method is more flexible, the second technique is better suited for multicast scenarios. The combination of resource reservation and media scaling seems to be particularly well suited to meet the varying demands of distributed multimedia applications.

Key words: Media scaling – Multimedia networks – Transport systems

1 Introduction

The dispute of guaranteed vs nonguaranteed communication is an unresolved argument in the multimedia community (as shown, for example, by recurring discussions at the first three International Workshops on Network and Operating System Support for Digital Audio and Video from 1990 to 1992). It is a repetition of the classic end-to-end argument: One group says that all mechanisms to cope with network bottlenecks should be included in the application; the other group says that only the underlying system is able to prevent network overload. In this paper, we propose a solution between the two extremes that offers both possibilities in an actual system. We favor this approach because different multimedia applications have different requirements on the network: there is virtually no way to recover from audio transmission errors so that the end user will not notice them. For everyday (consumer-quality) video, on the other hand, it is fairly easy to live with network flaws and even with slight delay variations.

HeiTS, the Heidelberg Transport System [6, 7], facilitates the transmission of digital audio and video from a single origin to multiple targets. The transport and network layer protocols of HeiTS, HeiTP [3] and ST-II [15] allow the client to negotiate quality-of-service (QOS) parameters such as throughput and end-to-end delay for multimedia connections. In its original form, HeiTS depends on some type of bandwidth allocation mechanism in the underlying network to provide a transport connection with a guaranteed QOS. Some networks such as FDDI (with its synchronous mode) and ISDN implement this reservation. Other networks such as Token Ring can be augmented with bandwidth allocation schemes [11]. However, not all kinds of networks support the reservation of bandwidth: as an example, Ethernet provides no guaranteed service at all due to the potential collisions of packets[1]. Hence, to use audio and motion video in such an "unfriendly" environment calls for additional techniques. When reservation is not available, audio and motion video should be transported on a best-effort basis. From the start, HeiTS has supported some kind of best-effort QOS [19] which, however, is only a less strict version of guaranteed QOS. In this best-effort approach, resource capacities are reserved, but at the same time statistically multiplexed, that is, the sum of the portions of bandwidth allocated to the individual sessions is allowed to exceed the total resource capacity. Best-effort service with no reservation requires a different approach, which can work, for example, in a dynamic feedback fashion. Here, the system monitors how well it currently ac-

Correspondence to: C. Halstrick

[1] For this reason, some "multimedia" solutions for the Ethernet use a 10BaseT hub and dedicate single Ethernet links to pairs of communication partners. This approach requires changes in the network infrastructure and still leaves unsolved the problem of conflicting uses of the dedicated links by multiple concurrent multimedia applications on the same machine.

complishes the audiovisual data transport from one end to the other, then correspondingly determines the amount of audio visual data it forwards. We refer to this technique as "*media scaling*."

Media scaling in different forms has been suggested and used in previous systems. Fluent, for example, bases its multimedia networking technology on a proprietary scaling scheme [16]. Tokuda et al. have developed a dynamic QOS management service, which is intended to be used in conjunction with scaling techniques [14]. Clark et al. with their "predicted service" approach also assume in their networking architecture that some form of media scaling exists [2]. Our approach is special and different in that it shows how to combine resource reservation and media scaling methods.

This paper discusses several implementation alternatives for media scaling in HeiTS. Section 2 surveys scaling methods, concentrating on digital video. Section 3 introduces two different scaling methods for HeiTS. Section 4 specifies the changes to protocols and interfaces in HeiTS required to accommodate scaling.

2 Scaling methods

Before describing the details of the HeiTS approach, we give a brief survey of scaling techniques. We assume that the reader is familiar with typical encoding schemes for digital media. "Scaling" means to subsample a data stream and only present some fraction of its original content. In general, scaling can be done at either the source or the sink of a stream. Frame rate reduction, for example, is usually performed at the source, whereas hierarchical decoding is a typical scaling method applied by the sink. Since in the context of this paper scaling is intended to reflect bandwidth constraints in the underlying resources, it is useful to scale a data stream before it enters a system bottleneck; otherwise it is likely to contribute to the overload of the bottleneck resource. Scaling at the source is usually the best solution here: there is no need for transmitting data in the first place if it will be thrown away somewhere in the system. Scaling methods used in a multimedia transport system can be classified as follows:

- *Transparent scaling* methods can be applied independently from the upper protocol and application layers, that is, the transport system scales the media on its own. Transparent scaling is usually achieved by dropping some portions of the data stream. These portions – single frames or substreams – need to be identifiable by the transport system.
- *Non-transparent scaling* methods require an interaction of the transport system with the upper layers. In particular, this kind of scaling implies a modification of the media stream before it is presented to the transport layer. For the distribution of media captured in real time, nontransparent scaling typically requires modification of some parameters of the coding algorithm. Stored media can be scaled by re-

In a multimedia system, scaling can be applied to a couple of different media types. Examples are video, audio, pointer device control streams, sensory information (e.g., data gloves).

For pointer device control streams or sensory information, scaling can in general be achieved by simply reducing the sampling rate. Bandwidth requirements of these streams are usually low compared to audio and video streams; therefore, performance gains achieved by applying scaling mechanisms are rather small.

For audio, scaling is usually difficult because presenting only a fraction of the original data is easily noticed by the human listener. Dropping a channel of a stereo stream is an example.

For video stream users are typically much less sensitive to quality reductions. Therefore and because of their high bandwidth requirements, video streams are predestined for scaling. The applicability of a specific scaling method depends strongly on the underlying compression technique, as will be explained in Sect. 2.2. There are several domains of a video signal to which scaling can be applied:

- *Temporal scaling* reduces the resolution of the video stream in the time domain by decreasing the number of video frames transmitted within a time interval. Temporal scaling is best suited for video streams in which individual frames are self-contained and can be accessed independently, such as intrapictures or DC-coded pictures for MPEG-coded video streams [9]. Interframe compression techniques are more difficult to handle because not all frames can be easily dropped.
- *Spatial scaling* reduces the number of pixels of each image in a video stream. For spatial scaling, hierarchical arrangement is ideal because it has the advantage that the compressed video is immediately available in various resolutions. Therefore, the video can be transferred over the network using different resolutions without applying a "decode → scale down → encode" operation on each picture before finally transmitting it over the network.
- *Frequency scaling* reduces the number of DCT coefficients applied to the compression of an image. In a typical picture, the number of coefficients can be reduced significantly before a reduction of image quality becomes visible.
- *Amplitudinal scaling* reduces the color depths for each image pixel. This can be achieved by introducing a coarser quantization of the DCT coefficients, hence requiring a control of the scaling algorithm over the compression procedure.
- *Color space scaling* reduces the number of entries in the color space. One way to realize color space scaling is to switch from color to gray-scale presentation.

Obviously, combinations of these scaling methods are possible.

Whether nontransparent scaling is possible depends strongly on the kind of data to be transmitted. For live video streams, it is easy to set all the coding parameters when an image is sampled at the source. For stored video, scaling may make a

The efficiency of a scaling algorithm strongly depends on the underlying compression technique. The format of the data stream produced by the coding algorithm determines which of the domains is appropriate for scaling. The following enumeration gives a short overview of the applicability of scaling to some state-of-the-art compression techniques.

- *Motion JPEG.* The distinguished feature of motion JPEG encoding (that is, the encoding of video as a sequence of JPEG frames [20]) is its robustness to transmission errors because of the independence of individual frames: a single error is not carried over from one frame to another. Obviously, temporal scaling is suited best for this compression technique, as any frame can be left out without affecting its neighbors. Applying a hierarchical DCT-based compression method on every picture [16] enables spatial scaling methods. However, few existing JPEG implementations realize this hierarchical mode.
- *MPEG.* Since MPEG [9] is a context-sensitive compression method, temporal scaling is subject to certain constraints. Every compressed video stream consists of a sequence of intra-coded, predicted-coded, and bidirectionally-coded pictures. Temporal scaling of an MPEG coded video stream can be realized by dropping predicted and bidirectionally coded pictures. Assuming an intra-picture is inserted every 9th frame, this leads to a scaled frame rate of approximately 3 frames per second [13].

The main improvement of MPEG-2 over the original MPEG scheme is the support for scalable media streams [3, 5, 14]. It facilitates spatial and frequency scaling as well as temporal scaling methods. MPEG-2 uses a hierarchical compression technique which enables the compressed data stream to be demultiplexed into three substreams with different quality. Scaling can be achieved by transmitting only some but not all of the substreams. This method is particularly useful for the discrete scaling approach described in Sect. 3.2.

- *DVI.* Just like MPEG, DVI [10] uses a combination of intra- and intercoded frames. Thus, temporal scaling is restricted in the same way, as described for MPEG-coded streams.
- *H.261 (px64).* The H.261 standard includes an amplitudinal scaling method on the sender side [17]. The coarseness of the quantization of the DCT coefficients determines the color depth of each image pixel. In addition to this, the intra-frame coding scheme, which is similar to the intra-coded pictures of MPEG, permits the easy use of temporal scaling.

3 Scaling in HeiTS

HeiTS is a rate-controlled transport system. For every data stream passing through a HeiTS connection, the system is informed about its message rate by means of an associated QOS parameter set. At the transport level interface, this rate is given in terms of logical data units (for example, video frames) per time period. HeiTS can use this information for monitoring target can inform the origin about the overload and cause it to scale down the stream. Once the overload situation has passed, the stream may be scaled up again.

A scalable stream can be seen as composed of various substreams. For a spatially scaled stream this representation can, for example, consist of one substream with all odd/odd pixels, one substream with even/even pixels, etc. As an alternative, one could use one substream for intra-coded frames and one or even several other streams for the remaining frames, which implies that there are streams of different degrees of importance. A splitting of MPEG video streams based on DCT coefficients has been described [12].

In the scaling implementation of HeiTS, individual substreams are mapped onto different connections, each with its own set of QOS parameters. The transmission quality can then be adjusted either with fine granularity within a connection (substream) or with coarse granularity by adding and removing connections (substreams). We refer to these approaches as *continuous* and *discrete* scaling. These approaches, together with a discussion of the monitoring functions needed, are discussed in the following subsections.

3.1 Monitoring

The prerequisite for any scaling mechanism is a function that allows the system to detect network congestion. For HeiTS, this can be achieved by monitoring two QoS parameters: *end-to-end delay* and *TSDU loss rate*.

3.1.1 End-to-end delay

In HeiTS, each logical data unit is represented by a transport service data unit (TSDU). Each TSDU has an expected arrival time, and a TSDU arriving later than expected indicates congestion.

There are several possibilities to define the expected arrival time of a TSDU. One could, for example, define its value simply as the actual arrival time of the previous TSDU plus the period of the message stream (that is, the reciprocal value of its rate). Alternatively, the arrival time of the first TSDU of the stream (or any other earlier packet) rather than the arrival time of the previous packet could be used as an "anchor" for the calculation. This helps to avoid false indications of congestions in cases where the previous TSDU happened to arrive early and the current TSDU has a "normal" delay.

HeiTS calculates the expected arrival time as the "logical arrival time" of the previous packet plus the stream period. The logical arrival time is the arrival time observed when bursts are smoothed out, that is, when early packets are artificially delayed (for example, in a leaky bucket fashion) such that the specified stream rate is not exceeded (see [19] for details).

3.1.2 TSDU loss rate

A problem arising when only monitoring end-to-end delay is

loss rate. In HeiTS, both parameters are monitored in parallel. To adjust the value of the threshold value for the delay HeiTS continuously compares the mean-end-to-end delay to the corresponding value of the mean loss rate.

The lateness/loss of a single TSDU should not immediately trigger the scaling down of a stream, because the congestion may only be short. However, if a sequence of packets is late (or some packets are missing because they were dropped due to buffer overflow), it can be assumed that the network is congested. In this case, the receiver initiates a scale-down operation.

In HeiTS, the mean values of loss-rate and end-to-end delay are calculated over a predefined measurement interval. The determination of the interval length depends on the network characteristics and, therefore, has to be based on heuristics.

3.2 Continuous scaling

A major issue with the scaling procedure is the responsiveness, that is, how rapidly the traffic adapts to the available bandwidth. We propose a scale-down scheme which consists of three stages.

- The first reaction to a congestion is to throw away excess or late packets. This usually happens within the network during a buffer overflow or at the receiver station that detects the lateness of a packet. An appropriate mechanism for lateness detection is included in HeiTP. Scaling by dropping packets is immediate and local, that is, it does not affect the sender, which continues to send at its full rate. Hence, scaling up can also be done very quickly by simply stopping to discard packets. As stated before, it makes sense not immediately to trigger the sender to scale down the stream, since the congestion may only be brief.
- When the number of late or lost packets exceeds a certain threshold, which can be defined heuristically, it is assumed that the congestion will last longer. In this case, the sender is triggered to throttle its traffic. As a first step, the sender reduces its sending rate – possibly down to zero. (Reducing the rate to zero makes no sense if all data are sent over only one connection. If continuous scaling is applied to one of several substreams, this substream may temporarily carry no data at all and the receiver will still receive information.) The connection, however, remains intact, along with its resource reservations[2]. This means that the resources can be temporarily used by other traffic, but the sender can scale the stream up immediately once the congestion is over.
- If the rate on a stream has been reduced to zero and the congestion is of a longer duration, that is, if several attempts of the sender to scale up the stream fail, the corresponding connection is terminated and all resources reserved for it are released. Since congestion typically occurs only at one bottleneck on the end-to-end connection (for example, on some subnetwork), the resources previously reserved on other subnetworks or nodes are made available for other connections. Scaling the stream up, however, requires the reestablishment of the connection, which takes some time.

This last step leads us directly to the discrete scaling approach which will be discussed in the next subsection.

The monitoring of a stream provides the receiving station with hints at congestion situations. This monitoring cannot, however, yield any information about the termination of a congestion. Assuming that the underlying network also does not give any explicit indication of congestions, the decision whether to scale up a stream must be based on heuristics. The only practical heuristic known is to scale up the stream when a certain time span after the previous scale down has elapsed.

A scale-up decision based on time spans can come either too early or too late. A scaling which is too late is not considered harmful if it happens within the range of a few seconds. If the transmission quality is temporarily reduced, a human user does not care much whether this lasts for 3 or 5 s. The effects of a scaling which is too early can be more severe. Scaling a stream up while the congestion situation is still present causes the receiver to trigger a new scale-down and, in the extreme case, an oscillation of the system. This implies an increased overhead for both end-systems and network and, additionally, can extend the phase of reduced quality longer than necessary.

To avoid oscillation, the scaling procedure of HeiTS scales up stepwise, as is done by other dynamic congestion control algorithms [1, 8]. After scaling down the stream, the sender transmits for a certain time span or a certain amount of data (for example, n packets for a fixed value of n) at the reduced rate. If after this period no scale-down message is obtained from the receiver, which means that, HeiTS could transfer the packets without any severe congestion, the sender increases its rate by some amount[3]. This procedure is continued until the maximum throughput for this stream is reached or until the receiver requests to scale down the stream again.

A simple example of the protocol machine for continuous scaling is shown in Figs. 1 and 2. The source entity consists of three stages: In the OK state, the source transmits the media stream in best quality. If a scale-down message is received from the sink, there is a transition to the DOWN state. The machine remains in the DOWN state until no further scale-down message is received for a certain time span t_{up}. In this case, the protocol machine goes to the UP state and tries to scale up the stream. If the maximum quality is reached the protocol machine returns to the OK state.

[2] Note that not only guaranteed connections but also best-effort connections in HeiTS may have resources reserved for them. However, the reservation for best-effort connections does not account for the worst possible case. Thus, in some situations the amount of reserved resources may not suffice, which will lead to congestions. On the other hand, best-effort conntions may temporarily use resources

[3] Note the difference between our scheme and the slow-start algorithm in TCP [8]. TCP's slow-start algorithm uses acknowledgements returned by the receiver to increase the traffic rate,

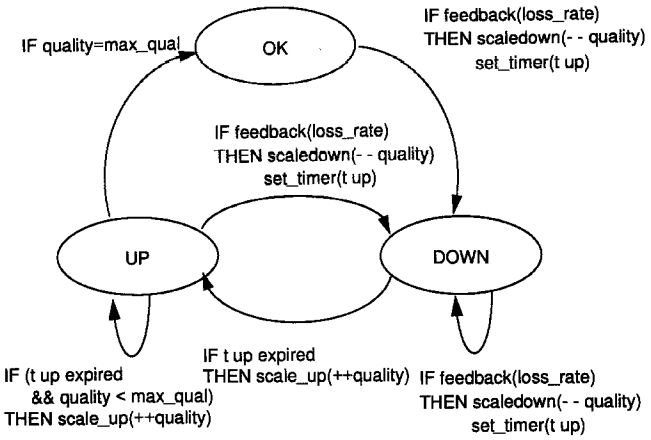

Fig. 1. State diagram source

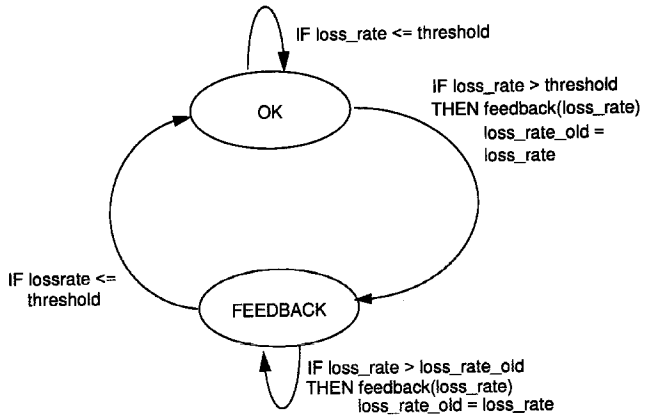

Fig. 2. State diagram sink

The protocol machine on the sink side consists of only two states: The OK state is left if the TSDU loss rate/delay exceeds the threshold value. During the transition to the FEEDBACK state, a scale-down message is sent to the source entity. The FEEDBACK state ensures that the sink entity waits at least for the time until the media stream is adjusted according to the last scale-down message, before a new scale-down message can be sent.

3.3 Discrete scaling

The advantages of the continuous scaling technique are that scaling can be done at fine granularity and that in principle only one connection is required per stream. There are, however, some problems with this approach because it does not take into account two special features of HeiTS.

- HeiTS supports multicast. This implies that continuous scaling may lead to the following problem. If a receiver triggers the sender to scale down the rate, all receivers from that point on get data at the lower rate, that is, a multimedia stream of worse quality. This approach is "all-worst" (or
- HeiTS supports different connection types. HeiTS has guaranteed connections for which all required resources are reserved in advance, and hence the requested throughput can be guaranteed. Additionally, HeiTS supports best-effort connections, in which no resources or only part of the resources required are reserved in advance; thus congestion is possible.

The discrete scaling technique discussed in the following is based on splitting a multimedia stream into a set of substreams, as described in the beginning of Sect. 3. This technique can be used in a multicast environment and supports different rates for different receivers. It works in an "individual best" (capitalistic) fashion.

For each of the different substreams a separate network layer connection is established. ST-II, the network protocol of HeiTS, in principle treats each of these substreams independently. However, the "stream group identifier" of ST-II can be used to indicate that several network connections belong to a single transport connection. The system can then try to achieve roughly the same delay for each of these network connections that facilitates reassembly of the substreams as packets reach the target with approximately the same transit time.

For establishing a set of substreams, an application specifies the percentage of data, which has to be transmitted to the receiver under any circumstance. If fewer data are transferred, a receiver cannot decode any useful information. These data are transferred over a guaranteed connection, if possible. If no guaranteed connections can be supported (for example, because there is an Ethernet in between), a best-effort connection is also used for this portion of the stream.

The rest of the stream is transferred over one or more best-effort connections. How many connections of this kind are required depends on the granularity of the data stream: Each part that provides a useful increase in quality is transferred over a separate best-effort connection.

Example: A video data stream is sent with 24 frames per second (fps). The sender decides that 6 fps have to be transferred under any circumstances to the receivers. These data are sent over the basic connection. The remaining 18 fps might be sent over two best-effort connections: 6 fps over the first and 12 fps over the second. In this example, the two best-effort connections have different throughput requirements. The video frames are then sent in the following order over the different connections:

1	2	3	4	5	6	7	8	9	...
bas	be_2	be_1	be_2	*bas*	be_2	be_1	be_2	*bas*	...

bas: sent over basic connection (guaranteed or best-effort)
be_1: first best-effort connection
be_2: second best-effort connection

If a receiver detects some congestion on any of these connections, it closes the least important connection (that is, be2 in the example). If we have a multicast connection, this disconnect does not necessarily imply a termination of the whole connec-

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.