Video Transmission Using the Available Bit Rate Service

By

Ronald Bollow

Submitted to the Department of Electrical Engineering
- Telecommunication Networks Group -

at

Berlin University of Technology
in Fulfillment of the Requirements for the

Diplomarbeit

Prof. Adam Wolisz
Supervisor: Prof. Henning Schulzrinne

Berlin 14. January 1997

Contents

1 Introduction

To transmit video data over ATM some services have been defined by the ATM-Forum. The constant bit rate service (CBR) reserves the bandwidth if available requested by the source as a fixed bandwidth for a virtual circuit (VC). This is done during connection establishment. Sources using this service are assumed to send at a constant bit rate. Since video sources are bursty, that is, they produce varying bit rates, the variable bit rate service (VBR) has been defined. For VBR connections, a guaranteed constant bit rate is determined during connection setup. But the source may vary around this rate according to the *Generic Cell Rate Algorithm* (GCRA). The GCRA is defined with two parameters: the increment (I) and the limit (L), and it maintains a *Theoretical Arrival Time* (TAT) for each cell. If a cell to be sent arrives at time t_a, GCRA checks whether $TAT < t_a + L$. If this condition is met, the cell is said to be conforming, that means, it can be sent immediately. After that, TAT is incremented by I. Thus, short bursts may exceed the cell rate negotiated at connection setup or, in other words, the guaranteed rate needs not to be as high as bursts due to scene changes or high motion scenes would require. The difficulty, however, is to choose the right rate at connection establishment.

These transmission services have the disadvantage of reserving fixed bandwidth during connection establishment with no possibility to change it without disconnecting. If the rate is chosen too high some link bandwidth will remain unused. This leads to the effect that new connections may be rejected although link bandwidth is still available. One solution is to enable the source to specify a minimum required bandwidth and letting the network provide more bandwidth to the source, if available, when the connection is established. This way, many more simultaneous connections can be provided by the network and thus the utilization will increase significantly. The Available Bit Rate service (ABR) employs this principle.

Because an ABR source is required to be able to decrease its sending rate at any time the ABR service class is originally designed for non-real-time applications. However, since video transmission becomes more and more important it would be desirable to use the ABR service also for video. This would be nice because ABR connections will probably be the future low-cost offers of network providers. Likely, there may be designed a new ABR-like ATM service for the special needs of video sources as proposed in [21]. However, we preferred to use an established standard. The problem is that video data can be buffered only a very limited time because the video decoder must have received all data fragments of a frame if the time to show the last frame is up. Otherwise no correct video sequence can be produced by the decoder. For that reason the video encoder itself might reduce its output rate if a rate decrease is required by the network. The basic idea to make this possible is to control the video encoder by the feedback from the network. A video encoder can adapt its sending rate to a given rate by changing the frame rate and/or the quality of the quantizer. We are going to set up a rate control mechanism which controls the frame rate of the encoder by the network feedback.

In chapter 2 we give a review of the ABR service as defined by the ATM-Forum. Chapter 3 describes the source models used for our simulation study. In the 4th chapter we introduce two approaches for the rate control mechanism. Chapter 5 finally presents the simulations we performed to evaluate ABR based video transmission. These simulations will show how the ABR mechanism manages bursty sources like video sources and how the proposed rate control works. Further we will get the utilization of the available link bandwidth and the end-to-end delay as well as the frame rate variation. At the end we will depict our impression of looking at a real video sequence transmitted over our simulated ABR channel.

2 The ABR Service

2.1 Service Category Definition

ABR is an ATM layer service category [17] for which the limiting ATM layer transfer characteristics provided by the network may change subsequent to connection establishment. A flow control mechanism (see next section) is specified which supports several types of feedback to control the source rate in response to changing ATM layer transfer characteristics. It is expected that an end system that adapts its traffic in accordance with the feedback will experience a low cell loss ratio and obtain a fair share of the available bandwidth according to a network specific allocation policy. In contrast with CBR and real-time VBR, cell delay variation is not controlled in this service, although admitted cells are not delayed unnecessarily. ABR service is not intended to support real-time applications.

On the establishment of an ABR connection, the end-system shall specify to the network both a maximum required bandwidth and a minimum usable bandwidth. These shall be designated as peak cell rate (PCR), and the minimum cell rate (MCR), respectively. The MCR may specified as zero. The bandwidth available from the network may vary, but shall not become less than MCR.

2.2 Service Parameters

Table 1 lists all parameters which are used to implement ABR flow control on a per-connection basis. Except ACR, Mrm and TCR, all these parameters are negotiated at connection establishment. The ACR is set initially to ICR and subsequently by the flow control mechanism. Mrm and TCR are constants.

The MCR should be the minimum bandwidth the application requires to work correctly respectively to work in a subjectively acceptable manner. This rate is always guaranteed by the network. For non-real-time applications this parameter mostly can be set to zero.

Label	Description
PCR	The Peak Cell Rate, PCR, is the cell rate which the source may never exceed.
MCR	The Minimum Cell Rate, MCR, is the rate at which the source is always allowed to send.
ICR	The Initial Cell Rate, ICR, is the rate at which the source should send initially and after an idle period.
RIF	The Rate Increase Factor, RIF, controls the rate at which the cell transmission rate increases.
Nrm	Nrm is the maximum number of cells a source may send for each forward RM cell.
Mrm	Mrm controls allocation of bandwidth between forward RM cells, backward RM cells, and data cells.
RDF	The Rate Decrease Factor, RDF, controls the decrease in the cell transmission rate in several cases.
ACR	The Allowed Cell Rate, ACR, is the current rate at which a source is allowed to send.
CRM	Missing RM cell count. CRM limits the number of forward RM cells which may be sent in the absence of received backward RM cells.

Label	Description
ADTF	The ACR Decrease Time Factor is the time permitted between sending RM cells before the rate is decreased to ICR.
Trm	Trm provides an upper bound on the time between forward RM cells for an active source.
FRTT	The Fixed Round-Trip Time, FRTT, is the sum of the fixed and the propagation delays from the source to the destination and back.
TBE	Transient Buffer Exposure, TBE, is the negotiated number of cells that the network would like to limit the source to sending during startup periods, before the first RM cell returns.
CDF	The Cutoff Decrease Factor, CDF, controls the decrease in ACR associated with CRM.
TCR	The Tagged Cell Rate, TCR, limits the rate at which a source may send out-of-rate forward RM cells.

Table 1: ABR parameter descriptions

2.3 RM Cells - The Network State Information Carrier

In the ABR service, the source adapts its rate to changing network conditions. Information about the state of the network like bandwidth availability, state of congestion, and impending congestion, is conveyed to the source through special control cells called *Resource Management* cells (RM cells). The flow of each virtual circuit (VC) is controlled separately using its own RM cells. Besides generic fields as there are ATM header, protocol identification and CRC, an RM cell contains the following fields:

- direction flag (forward/backward)
- BECN flag (BN)
- congestion indication flag (CI)
- no increase flag (NI)
- explicit cell rate (ER)
- current cell rate (CCR)
- minimum cell rate (MCR)

Furthermore, there are some fields that are not used for ATM-Forum ABR.

2.4 Network Feedback

There are defined two types of how the network gives feedback to the source

a) Binary feedback: If there is a congestion situation somewhere along the VC this is indicated to the source by setting the CI flag. A source receiving a RM cell whose CI flag is set shall reduce its rate by a predefined factor (RDF). If no congestion is indicated, the source may increase its rate by another predefined factor (RIF), unless the NI flag is set in a RM cell received. The NI flag is used to indicate that the network is approaching the congested state but is not actually congested. In this case the sending rate needs not to be reduced, but should not been increased.

b) Explicit rate feedback: Explicit feedback enables the network elements to compute concrete values for the workload that can be provided by the network. The source has to reduce its sending rate to the explicit rate (ER) proposed by the network. In addition, there is the possibility to provide explicit feedback information not just in the case of congestion situations but in case there is still some free network capacity as well.

The explicit rate feedback is more suitable for video transmission than the binary feedback because it is able to provide smaller cell loss rates. Also, it attempts to achieve the goal of max-min fairness for the source rates, while operating the bottleneck links efficiently. (Compare [11]) With an appropriate switch rate-allocation mechanism (e. g. EPRCA [19]) the computational overhead of explicit rate feedback is not very high. For these reasons, this study no longer deals with binary feedback.

Figure 1 shows how the ABR feedback loop controls the rate of an ABR source. The source regularly inserts RM cells into the data cell stream. They are processed by all network components along the VC path. Forward RM cells especially serve for switches to get information about the cell rate of any VC in order to compute the fair share. Furthermore, a switch may already indicate a congestion using the so-called Explicit Forward Congestion Indication (EFCI). This way, all network components behind this switch get to know (with short delay) that there is a congestion somewhere along the VC path. The destination returns the RM cell after setting the binary feedback flags and/or the ER field in accordance with its congestion state and its network interface capacity. On the backward path the switches likewise set the RM cell fields according to their congestion state and providing a fair share of the available bandwidth. If the source gets back the RM cell, it re-determines its allowed cell rate (ACR) using the information residing in the RM cell.

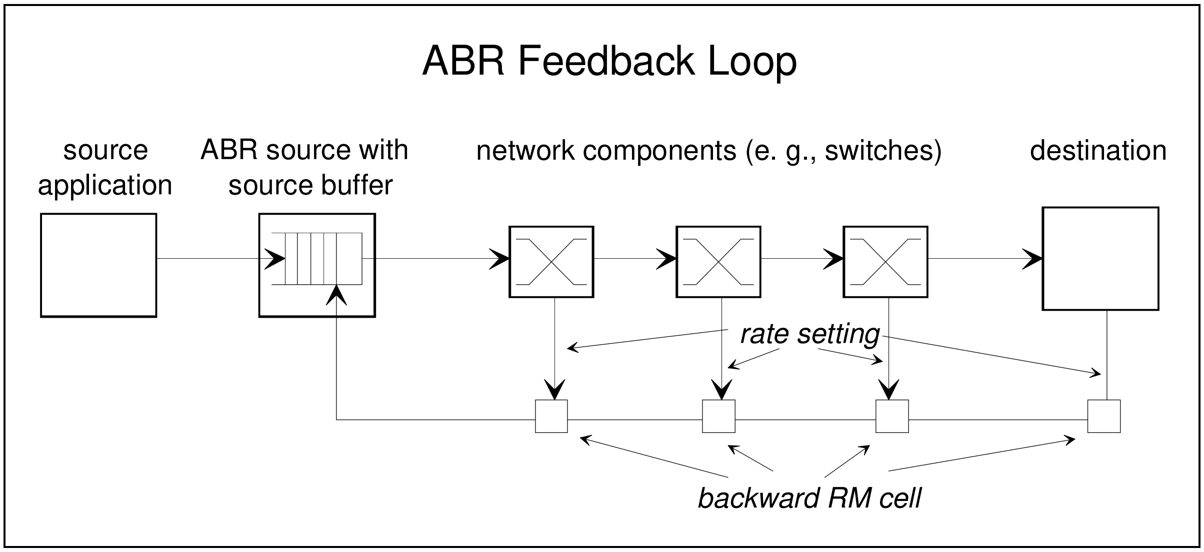

Figure 1: The ABR feedback loop with explicit rate feedback

Explicit rate switches normally wait for the arrival of an RM cell to give feedback to a source. However, under extreme congestion, they are allowed to generate an RM cell and send it immediately to the source. This optional mechanism is called backward explicit congestion notification (BECN). If a switch generates such RM cell, it has to set the BECN flag.

2.5 Source Behavior

In this section we present the most important source end system rules which an ABR source has to comply with. They show how the ABR source utilizes the certain parameters already described.

1. Sources should always transmit at a rate equal to or below their computed ACR. The ACR cannot exceed PCR and need not go below MCR.
2. At the beginning of a connection, sources start at ICR. The first cell is always a RM cell.
3. a) The source is required to send a forward RM cell after every 31 cells (Nrm-1). However, if the source rate is low, the time between RM cells will be large and the network feedback will be delayed. To overcome this problem, a source is supposed to send a forward RM cell if more than 100 ms (Trm) has elapsed since the last forward RM cell. This introduces another problem for low rate sources. In some cases, at every transmission opportunity the source may find that it has exceeded 100 ms and needs to send a forward RM cell. In this case, no data cells will be transmitted. To overcome this problem, an additional condition was added that there must be at least one other cell (Mrm-1) between forward RM cells.

   b) A waiting backward RM cell has priority over waiting data, given that no backward RM cell has been sent since the last forward RM cell. This way, backward RM cells are not unnecessarily delayed while the RM cells are not using up all available bandwidth.

   c) Data cells have priority in the remaining cells.

4. The rate allowed to the source is valid only for approximately 500 ms (ADTF). If a source does not transmit any RM cells for this duration, it cannot use its previously allocated ACR particularly if the ACR is high. The source should re-sense the network state by sending an RM cell and decreasing its rate to the initial cell rate (ICR) negotiated at connection setup. If a source's ACR is already below ICR, it should stay at the lower value (and not increase it to ICR).
5. If a source has sent CRM forward RM cells and has not received any backward RM cell, it should suspect network congestion and reduce its rate by the factor CDF. BECN cells generated by switches are not counted as backward RM cells.
6. a) If a received backward RM cell has a CI field of 1, the source should reduce its rate by ACR $\times$ RDF. The rate decrease is limited by MCR. If CI=0 and NI=0, the rate may be increased by RIF $\times$ PCR, but not higher than PCR. If NI=1 the source shall not increase its rate.

   b) A source should change its ACR to the new ER value. But any rate increase is limited to the factor of RIF $\times$ PCR. The rate decrease is limited by MCR.

   ACR is finally set to the minimum of the results from a) and b).

3 Modeling Video and Data Sources

3.1 Persistent and Non-persistent Sources

The simplest approach modeling a data source is a so-called *persistent* source. A persistent source is sending always at that rate it is allowed to. This model imposes the heaviest constraints on the network and is therefore very appropriate for testing the fairness and the throughput of the ABR service. We will use persistent sources for background loads.

A video source, however, needs to be synchronized and follows therefore a specific distribution. It does not care about any allowed rate. Thus data of such a source may get lost, unless the source distribution is appropriately controlled by the network.

3.2 An Analytical Source Model

Numerous video source models can be found in the literature. They may be divided into two groups. The first group [3, 14, 11] describes sources of video conference scenarios. There is only little motion in the sequences, and there are very few scene changes. The second group [9, 4, 11, 21] uses sequences from entertainment television, sports etc. Those are characterized by much motion and frequent scene changes. We think that the first group is appropriate to use future low cost connections which will probably provide the ABR service only, but not CBR or VBR. Therefore, we will concentrate on this group.

The number of bits/packets/cells per frame is the main parameter of most models. Together with the frame rate a video source is sufficiently determined by the distribution function of this parameter.

The ATM-Forum recommended a source model for ABR service evaluation [12] that is designed for asynchronous transmission. According to this model, a source has active and idle periods. During the active period it is sending packets of constant or geometrically distributed length. In ATM networks these packets are divided into cells. The pause periods between the packets are exponentially distributed. The number of packets in the active period is geometrically distributed.

In order to approximate this model to the characteristics of a video source we assume that one frame is transmitted during each active phase. Like most of the real sources the frame data is transmitted within one burst without significant pauses, that is, they are not spread out over the frame period. We further assume the packet length being 800 bytes (fixed). This is a realistic mean value for common video codecs, e. g. JPEG and H.261, used with the videoconference tool vic with picture contents moving slightly.

The number of packets per active period corresponds to the number of packets per frame. In [3] some different precise models for this parameter are presented. According to [3] as a first approximation we can use for it the gamma distribution. Nevertheless we decided to use the geometric distribution because it is recommended by the ATM Forum [12] for use in an ABR environment. So we get our results being comparable to those from other researchers in the ABR service field who will probably use the geometric distribution. On the other hand we used this model only for initial testing how the ABR service manages bursty sources. Section 3.4 introduces a trace driven source which will match real video sources much better.

3.3 Video Encoding

3.3.1 Overview

For transmitting video data several codecs have been developed. The most common are JPEG, H.261 and MPEG. The JPEG codec originally had been developed for transmitting images. Therefore it does not consider any inter-frame dependencies. The H.261 codec has been designed for video conferences over constant bit rate ISDN lines. It uses the temporal redundancy of video sequences, that is, the encoder output from a present frame depends on past frames. There are some adaptations necessary for using H.261 in a packet network, especially for handling packet losses (see next subsection).

The MPEG codec, however, is primarily designed for high motion sequences, e. g. movies. It also uses the temporal redundancy, but there are three frame types each for a certain type of redundancy encoding. The I frame is encoded in the so called intra-frame mode, that is, the encoder uses information only from the current frame. When encoding a P frame, or predicted frame, only the differences between the present frame and the last reference (I or P) are encoded. When encoding a B frame, the encoder performs bi-directional interpolation. It encodes differences between the present frame and the previous as well as the following reference (I or P) frames. When modeling a MPEG source the different size of these frame types has to be considered.

As mentioned already, we will concentrate on video conference type sources. That is why we employ the H.261 codec for evaluating our sources.

3.3.2 H.261 over RTP

The H.261 coding is organized as a hierarchy of groupings. The video stream is composed of a sequence of images, or frames, which are themselves organized as a set of Groups of Blocks (GOB). Each GOB holds a set of 3 lines of 11 macro blocks (MB). Each MB carries information on a group of 16x16 pixels: luminance information is specified for blocks of 8x8 pixels, while chrominance information is given by two “red” and “blue” color difference components at a resolution of only 8x8 pixels.

Blocks which have changed are encoded by computing the discrete cosine transform (DCT) of their coefficients, which are then quantized and Huffman encoded (Variable Length Codes). The result from the Huffman encoding is put into RTP packets. But, because of the hierarchical structure the MB is taken as the unit of fragmentation. Packets must start and end on a MB boundary, that is a MB cannot be split across multiple packets. Multiple MBs may be carried in a single packet when they will fit within the maximum packet size allowed and waiting until the next frame is encoded will not cause too much delay.

For synchronization, the sampling instant of the first video image contained in the RTP data packet is assigned to the RTP time stamp. However, if there are multiple frames encoded in a packet, the timing information of the H.261 frame header must be used.

The marker bit of the RTP header is set to one in the last packet of a video frame, and otherwise, must be zero. Thus, it is not necessary to wait for a following packet (which contains the start code that terminates the current frame) to detect that the frame is received completely.

H.261 uses the temporal redundancy of video to perform compression. This differential coding (or inter-frame coding) is sensitive to packet loss. The RTP protocol, however, has no mechanism for error detection or recovering. It is up to the application, that is coder and decoder, to handle the packet loss. After a packet loss, parts of the image may remain corrupt until all corresponding MBs have been encoded in intra-frame mode. There are several ways to solve this problem:

a) All frames are intra-frame encoded, and MB level conditional replenishment is used.

b) The intra-frame refreshment rate (132 is recommended) is adjusted according to the measured loss rate.

c) The decoder requests an intra-frame coded image refreshment after a packet loss is detected.

Since the videoconferencing tool we used supports only a) we will evaluate just MB level conditional replenishment.

3.4 A Trace Driven Source Model

Analytical sources like that described in section 3.2 are simple to implement, reproducible and most of their characteristics can be controlled easily. But, for most of real sources there is no model yet available that meets the real characteristics completely. Therefore we looked for another approach.

A source model that is based on a trace of a real video source is actually close to reality. But we have to guarantee that this model is representative. To get this we made a long time trace from television. As mentioned already, we chose a broadcast of video conference type. Moreover, we traced about 30 minutes from the German television talk “ARD-Presseclub” broadcasted on 21. July 1996. This talk was performed by 6 actors and with absence of any audience. Generally the camera focused on one actor. The 30 minutes trace duration ensures that there are no short time dependences (the mean scene duration is a few seconds) and consequently this trace should be representative for video conferences and talk or lecture distributions. According to German television system the talk is recorded with PAL. As frame grabber we employed a SunVideo card working in an UltraSparc 1 workstation.

Since a video conference frame sequence typically changes little from frame to frame, it would be desirable to encode the changes only, in order to reduce the amount of data per frame. The conditional replenishment feature used by certain video coders can do this. For our trace we used a codec that supports conditional replenishment, namely the H.261 codec. It is preferred for video conference sources, whereas the MPEG codec is to be used for high motion videos [11].

To get the trace we established a connection using the Real-time Transport Protocol (RTP). The sending application *vic* encoded the frames taken from the video grabber card and put it onto the network. Another workstation traced occurrence time and size of these RTP packets using *rtpdump*. The size includes all headers except Ethernet. This is because all these headers have to be transmitted over the ATM network, say, with *IP over ATM*. Each IP packet is fragmented in order to fit into the 48 bytes data area of the ATM cells. Assembling all cells that belong to a particular frame we have now a base for building a video source.

3.5 Interpolating Frame Rates

A video source model should have the ability to represent any frame rate. This can be done by linear interpolation from the trace. But the conditional replenishment feature causes the cell rate c to be a non-linear function of the frame rate f, because varying interframe gaps lead to varying numbers of MBs that have been changed. For this reason, one trace is not sufficient. We decided to take four traces, with 3, 6, 12 and 24 frames per second[1]. Table 2 presents the average frame size of the video sequence mentioned in the previous section for these four frame rates with the H.261 quality level being equal to 10. Note that there is a significant difference in the frame size. The interpolated $c = \mathrm{f}(f)$ graph of these traces with the same quality level setting is presented in figure 2. However the influence of the frame size on the frame rate depends on the quality setting. In Figure 3 it can be seen that better quality setting causes a stronger influence of the frame rate on the frame size.

frame rate f [f/s]	average frame size [kByte/frame]
3	6.4
6	5.4
12	4.9
24	4.0

Table 2: Frame size dependence on the frame rate for a H.261 coded low motion video

[1] The traces are available via `ftp://ftp.fokus.gmd.de/pub/step/vidarch/*`.

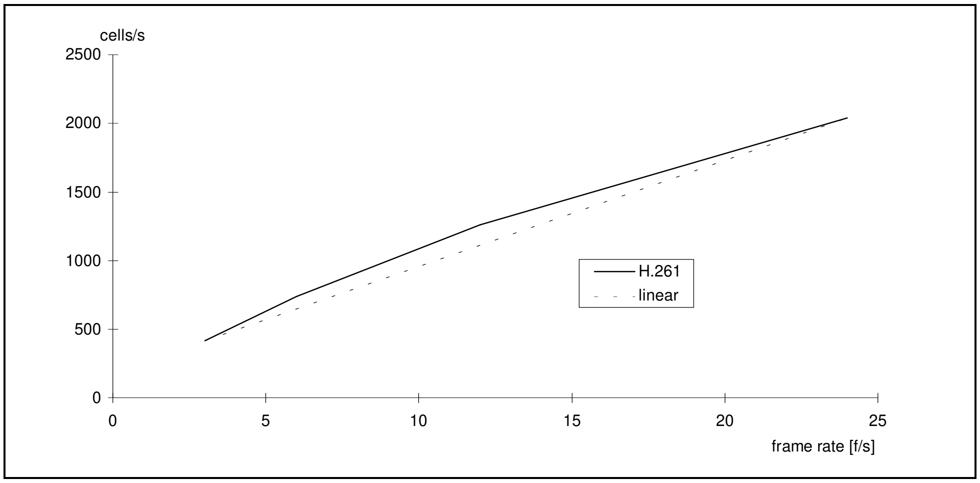

Figure 2: Cell rate c [cells/s] as a function of frame rate f [f/s] of a H.261 coded 30 min. videoconference

In particular we implemented our trace driven source model as follows. Each time a new frame is to be encoded the number of cells n_c in this frame is demanded from the trace database. At this point we know the current time t (by adding the duration of all frames until now) and the frame rate f of the new frame (assigned to the source, e. g. by the rate control). Now we take those two traces which have a frame rate next to f, e. g., if f=10 we take the trace with 6 f/s and that with 12 f/s. Then we look up in both traces that frame having a time stamp next to t. This is to synchronize the traces. The n_c of the frames looked up in the previous step are the base for interpolating linearly to get n_c of the frame that is sent next.

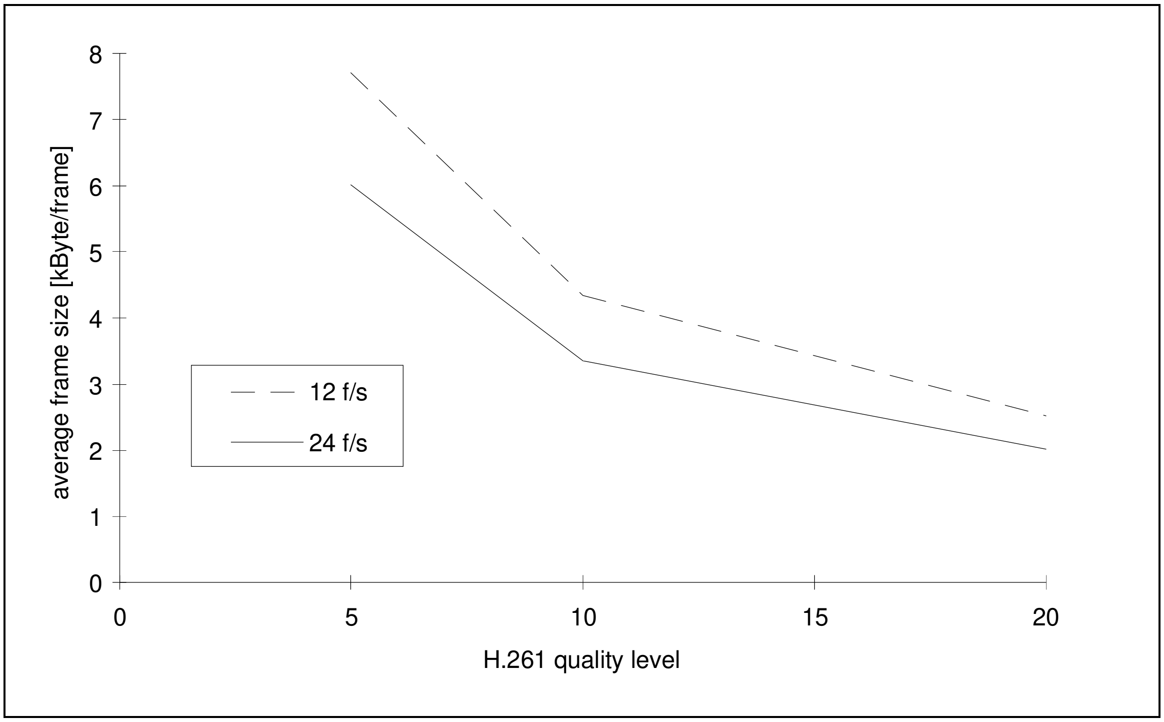

Figure 3: Frame size as a function of the H.261 quality level setting (higher value means lower quality) from a videoconference type sequence grabbed with certain frame rates

Figure 4 shows an example for the synchronized lookup of n_c. There are three time lines, the time lines of two traces and that of the interpolated source. For simplicity, we assume that the frame rate of the interpolated source is always between that of these two traces. Each marked point at the time lines represents a frame time stamp, that is the instant at which the frame is started encoding. The arrows indicate what n_c values (from which frames) are used as base for interpolating n_c of the frame they are pointing to.

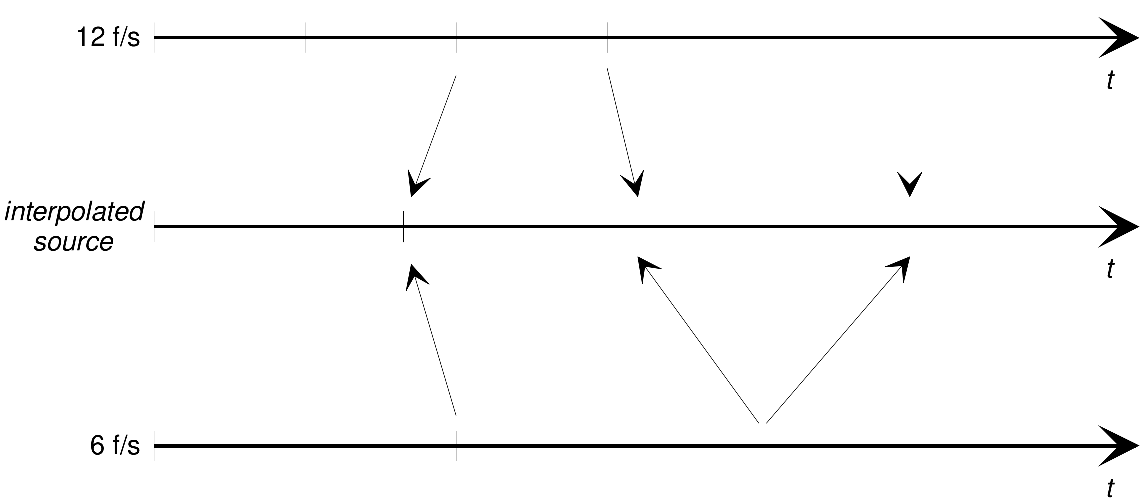

Figure 4: Example for the synchronized lookup of n_c to get base values for the interpolation

4 Building a Rate Controlled Video Source

To transmit video over an ABR connection, we have to control the output rate of the video encoder by the network feedback. However, it cannot be controlled directly but only through encoder parameters. Altering the quality parameter (quantizer step size) is often preferred for rate controlled video sources [21,11] because it provides relatively fine-grained tuning of the video bit generation rate. Furthermore, small changes in the quality parameter are barely visible to the human eye. Nevertheless, for low rate video transmission the rates reachable with quantization adjustment may not small enough for a perceptual quality. However, frame rate adjustment will be critical for ABR, for especially low frame rates lead to long delays in re-adjusting the coder. That is why we have focused on controlling the frame rate rather than the quality parameter.

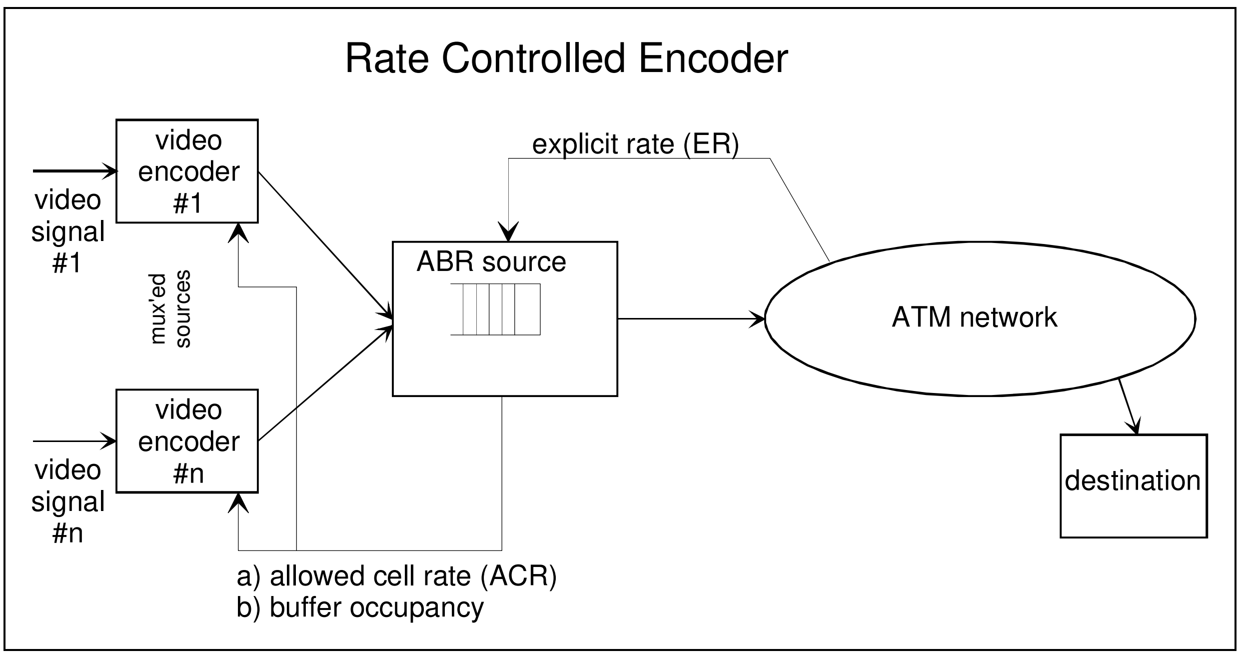

Figure 5: Encoder rate control by the network

The problem is now to find an algorithm which updates the frame rate using the network feedback. We investigate two approaches. The first uses the cell rate proposed by the network (ACR) directly for determining a new frame rate (section 4.1), the second utilizes the occupancy of the source buffer which is filled by the encoder(s) and serviced with ACR (section 4.2).

4.1 Direct Video Encoder Rate Control (using ACR)

The direct rate control is based on estimating the cell rate to frame rate ratio ρ of the average interval T_a for each new frame. Thus, with a given cell rate an appropriate frame rate to be used for the next frame can be computed. We take all frames which started within T_a before the current frame start. From these frames the averaged frame rate and cell rate are computed in order to get ρ for this interval

$$\rho = \frac{\frac{1}{T_m}\sum_{\forall m} n_c}{\frac{1}{n_m}\sum_{\forall m} r_f} \qquad (1)$$

T_m - actual interval taken by the averaged frames ($\leq T_a$)

n_m - number of frames within T_m

m - frame index within T_m

With this approach the encoder will be adapted very close to ACR. But this leads to a high variation of f. Moreover, a high frame rate variation will make the video transmission unacceptable for human users. In section 5.4 we present a simulation example of the indirect approach. For these limitations we searched for another mechanism which changes f slowly unless the source buffer is threatening to overflow.

4.2 Indirect Video Encoder Rate Control (using buffer occupancy)

The ACR based encoder rate control has among others the following disadvantages: It considers neither the history of the connection nor the current encoder output rate. This causes the above mentioned very large and fast variations of the frame rate. We propose therefore an indirect rate adaptation mechanism which is based on the occupancy of the source buffer (see figure 5).

4.2.1 When to Update the Rate

While the rate of the ACR controlled encoder is updated with each arriving backward RM cell the update interval for the indirect approach is independent of RM cell arrivals. However, the cell rate cannot be changed while a frame is being encoded. Therefore, a frame based update scheme should be used.

For simplicity, we trigger signaling new buffer occupancy values to the encoder by the RM cell arrivals. Since the frame size in our trace is between 40 and 220 cells and *Nrm* is recommended to be 32, between 2 and 7 RM cells are received during each frame. Thus, between 2 and 7 buffer occupancy values are available for each rate update. We use the first one occurring after a frame is transmitted completely by the encoder to the ABR source. This way, we do not need any control signaling between the encoder and the ABR source (both are located in the source end-system).

If the rate is set to a very small value, the delay until updating it next time is quite long; probably some bandwidth will remain unused. Therefore we specified a minimum frame rate $f_{\min}$. Each time the rate is computed to be equal or less than $f_{\min}$, it is set to $f_{\min}$, except if it is computed to be zero (encoding interrupt). In this case it is set to zero (nothing is sent), and after each multiple of the retry timer t_r (until $f > 0$) the frame rate is recomputed based on the latest buffer occupancy value and the minimum frame rate $f_{\min}$.

4.2.2 Rate Adaptation Mechanism

A rate adaptation mechanism based on buffer occupancy should have the following properties:

1. The frame rate f is computed recursively using a rate change factor RCF:

   $f_{(n)} = f_{(n-1)} \cdot RCF$ $\quad$ n - current frame number

2. If the source buffer has the target buffer occupancy b_t, f is not changed, otherwise f is changed in such a way that the current buffer occupancy b will approach b_t.
3. If b is near the maximum $b_{\max}$, f is reduced quickly; if $b \geq b_{\max}$, f is set to zero.
4. If property #3 is not met, f is changed slowly.
5. The variation of f is as small as possible.

The simplest function which is able to provide properties 2, 3 and 4 is a function of the form

$$RCF = \mathrm{f}\left(\frac{1}{b}\right) \tag{2}$$

The variable of this function is the deviation b-b_t normalized onto $b_{\max}$-b_t. In order to meet properties 2, 3 and 4 exactly we introduced a so-called *slope factor* which is denoted by α. It provides a single parameter that determines the slope of RCF while fulfilling both conditions, $RCF=1|b=b_t$ (property #2) and $RCF=0|b=b_{\max}$ (property #3). The resulting rate change function is

$$RCF = \frac{\alpha}{\alpha - 1}\left(1 + \frac{1}{(\alpha - 1)\dfrac{b - b_t}{b_{\max} - b_t} - \alpha}\right) \tag{3}$$

Figure 6 shows three samples of this RCF for b_t=0.5 $b_{\max}$.

When b reaches $b_{\max}$, the encoder has to maintain the old rate for the current frame. Because of this classical delayed feedback problem we have introduced a safety zone b_s, that is, the actual buffer size is $b_{\max}$+ b_s. The required extent of b_s mainly depends on how many multiplexed sources are using the same source buffer and which characteristics these sources have. We found out it empirically for our simulations.

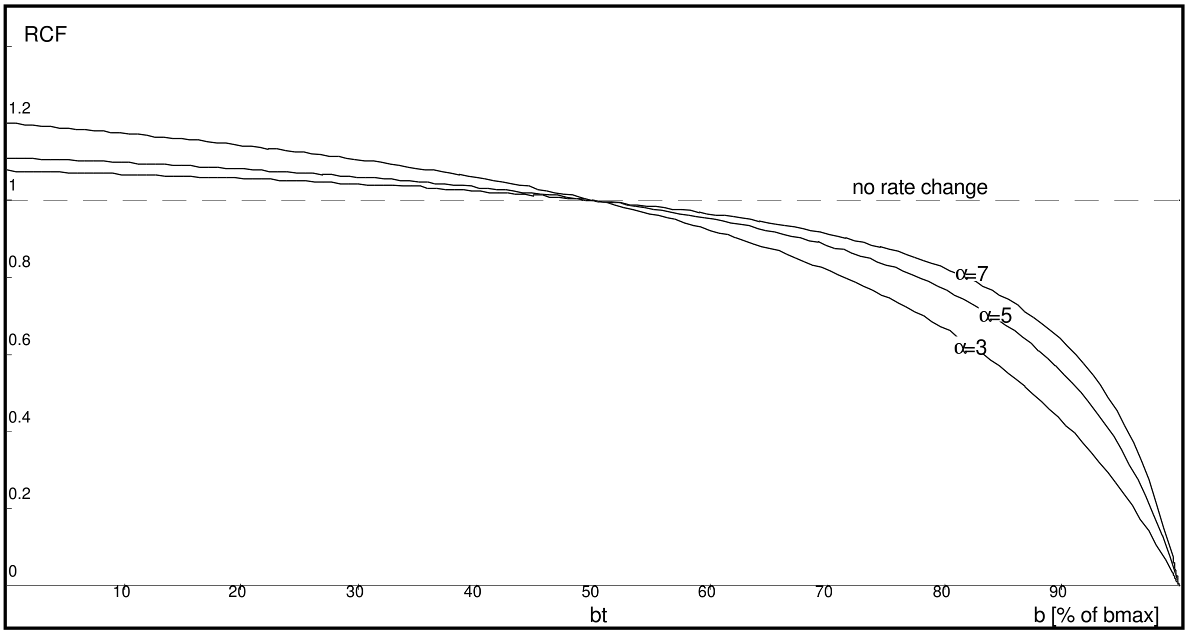

Figure 6: Rate change function RCF as a function of the buffer occupancy b for b_t=0.5 b_{max}

4.2.3 Damping Occupancy Oscillations

The rate adaptation function in equation (3) still has a strong disadvantage. If b is high, r will be reduced as long as $b > b_t$ even if b already decreases quickly. This together with the reverse effect (r is still raised although b increases quickly) leads to a large oscillation of the buffer occupancy b and to a corresponding rate oscillation. On the other hand r may be reduced not enough quickly if b increases very quickly just because it is not very high.

This problem forced us to put an additional "damping" component into the rate change function. The duty of this component is to correct the RCF considering any change of b since the last rate update. This rate change, however, is normalized onto $b_{\max}$ in order to prevent this correction from being overproportional. For weighting the damping component we introduced the damping factor β. Equation (4) shows the resulting formula for an updated frame rate $f_{(n)}$. Herein $f_{(n)}$ corresponds to $b_{(n)}$ (analogue for n-1).

$$f_{(n)} = f_{(n-1)} \cdot \frac{\alpha}{\alpha - 1}\left(1 + \frac{1}{(\alpha - 1)\dfrac{b_{(n)} - b_t}{b_{\max} - b_t} - \alpha} + \beta \frac{b_{(n-1)} - b_{(n)}}{b_{\max}}\right) \tag{4}$$

Our simulations showed that this additional component reduces the buffer occupancy oscillations and consequently the rate oscillations. Thus, the required buffer size is less than without it, there are fewer cell losses, and the average frame rate is higher. But caution is required in the use of β. If it is set too high, there will be other oscillations with their amplitude being equal or even larger than without this component. Useful values range between 0 and 1.

5 Simulations

5.1 Simulation Objectives

The basic objective of our simulations is to find out whether the available bit rate service defined by the ATM Forum is applicable for transmitting video with an appropriate quality of service. To achieve this we had at first to investigate how the ABR feedback control manages non-persistent ATM sources at all (section 5.3). For that purpose we used the analytical source model described in section 3.2. It is made similar to real video and is scaleable in a wide range. The only characteristic regarded at this source is the statistical distribution of the cell occurrences. The simulator used here does not know about frames and video codecs. Thus it will show us the utilization of links used by ABR video connections, the share of these connections in link bandwidth and the cell losses due to buffer overflows. In addition, we will get to know something about the cell rate variations.

The next step (section 5.4) is to use rate controlled video sources and to test our rate control mechanism described in chapter 4. This will give us frame level results as average and variation of the frame rate as well as end-to-end delay. Furthermore, we will see the extent of frame losses and/or encoding interrupts initiated by the rate controller. Of course, the link utilization and the share of link bandwidth will be seen, too. Since this simulation shall work on frame level we used here the interpolated trace driven source model described in section 3.4 and 3.5. As result we will get the input for our test of acceptability for human users (section 5.5).

5.2 Simulation Environment

5.2.1 Simulation Configurations

There are some configurations proposed in the literature which are useful for simulation of ATM networks. Configuration A (figure 7) [7] is used in order to investigate the behavior of a network with multiple ABR controlled video connections. The sources are denoted by *S*, the destinations by *D*. The video sources, S1 (wide area) and S2 (local), are located at different distances, that is, delay from the switch. The persistent source S3 makes the link is always utilized, even if the sum of the instant video source rates is low.

For investigating the fairness at the presence of one video source in a persistent sources network configuration B (figure 8) [9] was used. Here the VC to transmit video data (S1-D1) runs over three links each of them background-loaded by a persistent source.

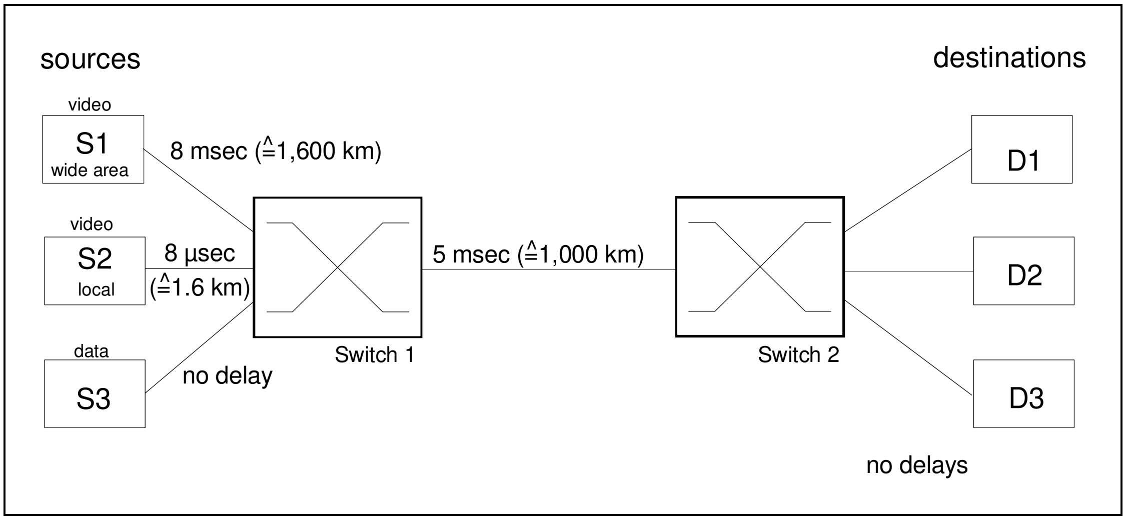

Figure 7: ABR Simulation - Configuration A

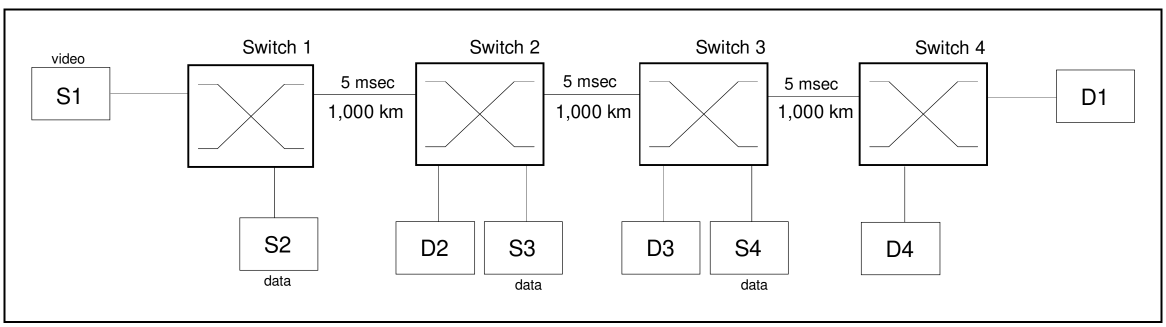

Figure 8: ABR Simulation - Configuration B

Each non-persistent ABR source (video) may be either a single video source (configuration A1/B1) or three or eight multiplexed, identically distributed but time-staggered video sources (configuration A2/B2). These sources constitute a single ABR source, that is, they are filling the same source buffer (compare figure 5). Although the multiplexing of video sources may take away some intensity of certain video source characteristics (for instance the rate variation) we decided to use them anyway because this will probably be a common scenario in future ATM networks.

5.2.2 The ABR Simulator

The simulator of the ABR mechanism consists of three parts, the sources, the destinations and the switches. They are all together taken from [18]. Here is a brief description of these parts.

The ABR source simulator has to set up the connection parameters, to send out the forward RM cells and to maintain the ACR based on the content of the backward RM cells. Because the Traffic Management Specification [17] was under development in the ATM-Forum, the source had to be modified in conformity with the latest version of the specification. Certain parameters (ADTF, CRM, CDF, TCR) were added to the source. Furthermore, the rate increase/decrease mechanism was redefined. The rate changes are not longer dependent on Nrm, and the rate increase depends on PCR.

Figure 9 shows the difference between the old and the new ABR source specification. The transmission rate (measured at the destination) of an on-off source (S1) in configuration B is presented. As can be seen the additional parameters (actually the underlying mechanisms) make the cell rate variation much smaller. Note that, due to ADTF, each burst starts at ICR for the new ABR whereas the rate of the old ABR (except within the first - transient - burst) highly exceeds the fair share of 50%.

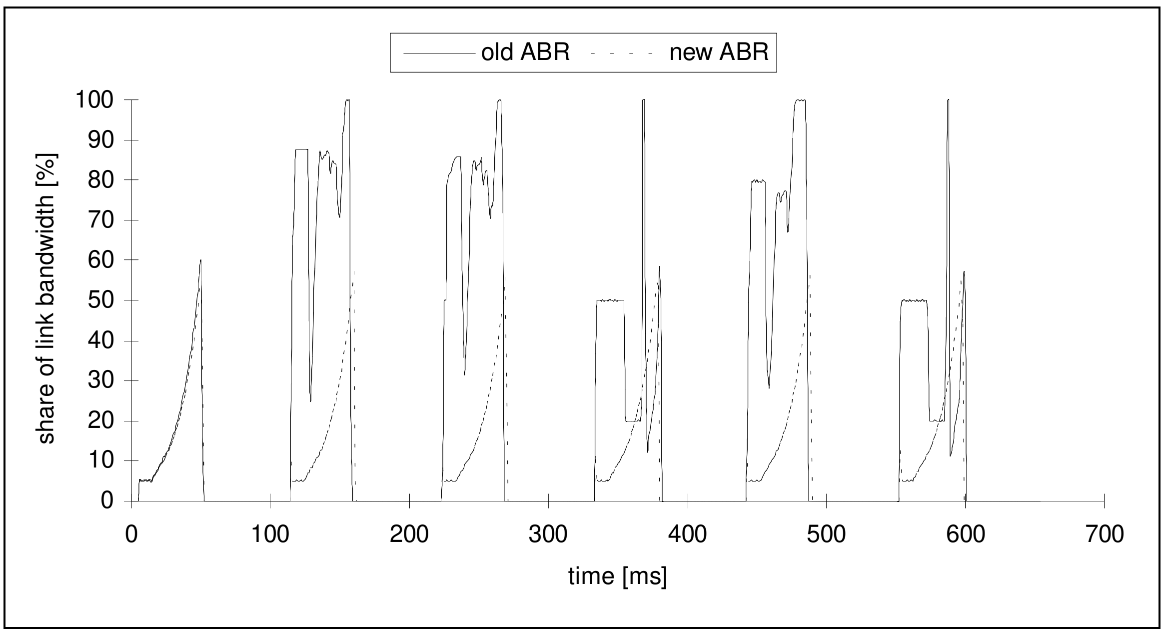

Figure 9: Cell rate of an on-off source using the older and newer specification version with configuration B (source S1) at 155 Mb/s link bandwidth

The ABR destination model is taken completely from [18]. The only task of this module is to reply the forward RM cells. Therefore no changes were necessary due to the change of [17].

Special attention should be paid to the switches. They have to modify the RM cells according to their congestion state. There are currently certain switch mechanisms proposed [19]. These mechanisms compute the explicit rate put into the RM cell. We used the Enhanced Proportional Rate Control Algorithm (EPRCA) [16], since it provides good fairness and requires not too much computational overhead. The EPRCA scheme computes a mean allowed cell rate (MACR) using exponential weighted averaging. The fair bandwidth share of each connection is then set to a fraction of this calculated MACR if necessary, that is, if the current cell rate indicated in the RM cells is larger than the fair share.

If nothing other is said, all ABR parameters are set to their default values recommended by the ATM-Forum.

5.3 Simulations with Analytical Sources

As mentioned above our analytical sources do not know about any allowed cell rate, that is, they are *not* controlled by the network feedback. For that reason a huge source buffer is recommended in order to reduce the losses. We decided to limit the buffer size to 10,000 cells. A full buffer thus takes about 5 sec of a video sequence according to the trace described in section 3.4. But the buffer will be full mostly in the case of a big frames. Consequently, the mean time spent to empty the buffer would be less.

All links provide a bandwidth of 45 Mb/s. The simulation time is 1.0 sec which is much more than the transient period. According to the simulations in [18], the MCR is set here to 1/100 of the link bandwidth. This value corresponds to about 40 cells per frame. According to [3], less than 10% of all frames in a video sequence are smaller than 40 cells. Therefore, it may be a good value in order to reject connect requests whose acceptance by the network would probably not be suitable.

A general result of this simulation is that the utilization of the links is, except during the transient period, almost always greater than 95%.

5.3.1 Using Configuration A

Table 3 shows the simulation results with configuration A. The number of cells per frame n_c is the only parameter that is varied. According to [3], a value of n_c=130 can be taken as close to reality for video conference scenarios. In order to increase the load, simulations with a higher n_c are made, too. Thus, in addition, those frames can also be taken into consideration that have a very large deviation from the mean and therefore cannot be produced by the geometric distribution.

average n_c	average transmission bandwidth [%]	
	local	wide area
130	3	3
400	10	10
1,000	18	17.3

Table 3: Average transmission bandwidth (fraction of link bandwidth) with configuration A

The column "average transmission bandwidth" shows the share of both video sources (the wide area and the local) in the link bandwidth. There is no data lost but only because the load produced by the application is significantly less than the fair share in the link bandwidth.

At n_c=400 and n_c=1000 the rate (especially that of the wide area source) oscillates aperiodically (up to 70% of link bandwidth). This causes the link utilization to take values around 50-70% of the link bandwidth for some measurement intervals. Moreover, it implies a high risk of buffer overflow and that at only 18% maximum load share!

5.3.2 Using Configuration B

Table 4 shows the simulation results with configuration B. The column "packet loss rate" contains the video source buffer overflow (averaged on the entire simulation time). This value considers all video packets that are lost or received incompletely due to source buffer overflow as a fraction of the entire number of packets sent during simulation time.

average n_c	average transmission bandwidth [%]		packet loss rate [%]	
	single	8x mux'ed	single	8x mux'ed
130	-	22	-	0
400	12	41.7 (79)	0	51
1,000	18	-	0	-

Table 4: Average transmission bandwidth (fraction of link bandwidth) and packet loss rate with configuration B1 (one single video source) and B2 (eight identically distributed, multiplexed, time-staggered video sources constitute a single ABR controlled source)

By the mean the fair share of the link bandwidth (here 50%) is not reached by the video source even if the link is fully utilized (compare source load produced by the application given in parenthesizes). Although the load is almost always less than the fair share data is lost. This is because of periodic rate oscillations with values between 40 and 60% of the link bandwidth. For eight sources at $n_c = 400$, e. g., the period of this oscillation is about 70 msec.

5.4 Simulations with Trace Driven Sources

5.4.1 Presumptions

As shown in the previous section the ABR mechanism works well at the presence of non-persistent sources. Thus we now concentrate on one configuration. We decided to use configuration A since it will show the correlation between multiple non-persistent sources. Furthermore, it will show how the connection quality depends on the source delay.

As mentioned in chapter 4 the direct encoder control mechanism causes strong variations of the frame rate. Figure 10 shows the differences in some quality parameters as an example that has been simulated. Especially the average rate change speed (MDT) values indicate very fast rate changes with direct rate control. Only the end-to-end delay has worse values with the indirect control, but at such a high level that this configuration (A2, 8 mux'ed sources, 5,000 cells buffer size) will be not useful for interactive video transmission anyway. Thus, the simulation series described in the following subsections is performed with indirect rate control.

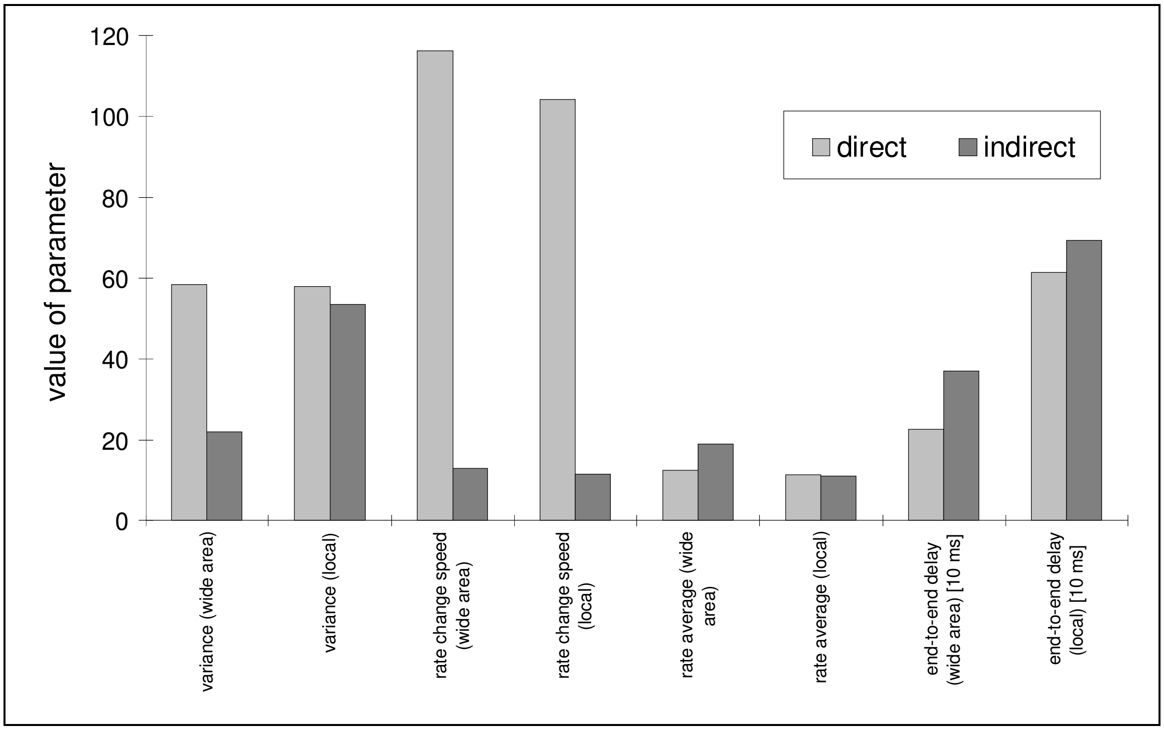

Figure 10: Chart of differences in some quality parameters between direct and indirect control with configuration A2, 8 mux'ed sources and 5,000 cells buffer size

The source buffer size is varied because we want to know the minimum possible buffer size. The safety zone b_s due to the delayed feedback problem is set to 500 cells. The target buffer occupancy b_t is set to $b_{max}/2$. This setting is determined empirically by short initial test simulations.

Since we used the controlled interpolated trace driven source here we can no longer take n_c as load parameter. Rather we varied the link bandwidth (same for all links) and the number of multiplexed video sources at each non-persistent ABR source (one, three or eight multiplexed video sources).

We chose link bandwidths B of 12 and 20 Mb/s. These values are much less than the bandwidth an ATM network may provide. They should be understood as remaining bandwidth from other CBR or VBR connections. These small values are chosen in order to force the video sources sending not continuously at the maximum frame rate. Thus we can see how our rate control mechanism works. The results presented in the following subsections are altogether from simulations with $B=12$ Mb/s. There are no major differences to the simulations with $B=20$ Mb/s. All quality parameters are of course somewhat better with $B=20$ Mb/s.

We set MCR again to $B/100$ (compare section 5.3). But, to achieve better video connection quality a high MCR would be useful. Therefore, we experimented a value of $B/10$ as well. This is about 40% more than a single video source demands. The ICR, however, must not be adjusted isolated from MCR. Since ACR is set to ICR if the ADTF timer expires, ICR determines the amplitude of cell rate oscillations, too. We set it to $B/20$ for MCR=$B/100$, and to $B/5$ for MCR=$B/10$.

5.4.2 General Results

A general result of our simulation is that the quality of video connections with a long delay between the ABR source and the connected switch (wide area) is throughout better than that of connections with a short delay (local). This is somewhat surprising since long delays are generally assumed to be bad. The following subsections will show this in detail.

A further general result is that the link utilization is independent of buffer size and encoder rate control mechanism. It is throughout very high (>95%). This verifies our result from simulations with analytical sources (see section 5.3).

Moreover, we experienced no significant influence of MCR or ICR on any connection quality parameters.

5.4.3 Frame Rate Behavior

Figure 11 and 12 show the frame rate as a function of time for the same time window of two simulations with identical parameter settings. The only difference is the source buffer size. With a smaller source buffer the rate decreases are more frequent and have lower minimum values.

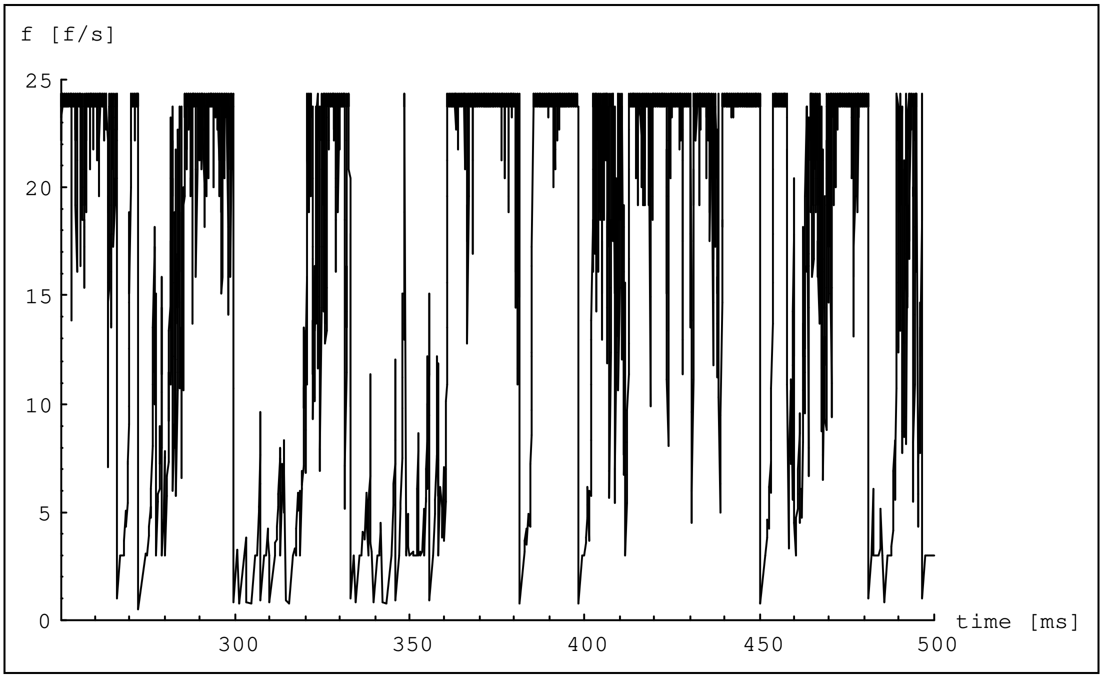

Figure 11: Time line extract of the frame rate for 8 mux'ed sources (configuration A2) with a buffer size of 2,000 cells

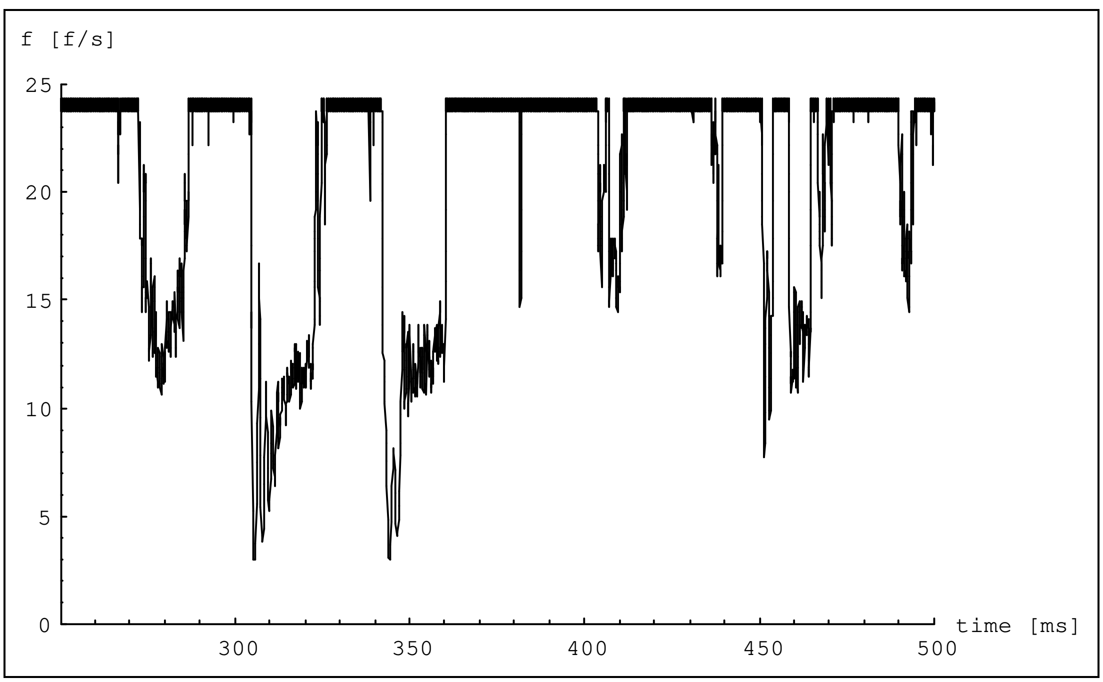

Figure 12: Time line extract of the frame rate for 8 mux'ed sources (configuration A2) with a buffer size of 10,000 cells

Figure 13 presents the average frame rate. Note the differences between the wide area and the local source. While the rate of the wide area source decreases at small buffer size the rate of the local one remains at its low level. There is a slight optimum near $\beta = 1$.

Parameters that characterize the frame rate variation are suitable to evaluate the acceptability of a video connection quality to human users. The *frame rate variance* describes the range of the frame rate. Another rate variation parameter is the derivative which describes the speed of rate variations. We determined it as follows

$$MDT = \frac{1}{T} \sum_{\forall n} \left| \frac{\Delta f}{\Delta t} \right| \tag{5}$$

MDT - Mean Derivative by Time

Δt - frame duration

T - total time

n - frame index

The *MDT* averages the rate changes over time from frame to frame as non-negative values, independent of whether the change is an increase or decrease. In contrast with the variance, *MDT* evaluates the speed of rate changes, not only the “diffusion” of the rate values.

The variance (figure 14) and the rate change speed *MDT* (figure 15) quantify the frame rate variation seen in figure 11 and 12. The behavior of *MDT* differs from all other characteristics, since here the local source looks better than the wide area source (see figure 15a). As can be seen in figure 15b it even depends on β whether the wide area source is better or the local one.

The dependency of the frame rate behavior on the number of multiplexed sources is shown in figure 16. As can be seen there, the differences between the wide area and the local source are the larger the more sources are multiplexed. Note that in our configuration for five or more multiplexed sources the average cell rate produced by the multiplexer is larger than the fair share. But, due to burstiness, the frame rate also varies with less multiplexed sources. Moreover, the number of multiplexed sources here determines what share of the entire link bandwidth do the video sources take. Without multiplexing, e. g., the share is 15% (both video sources).

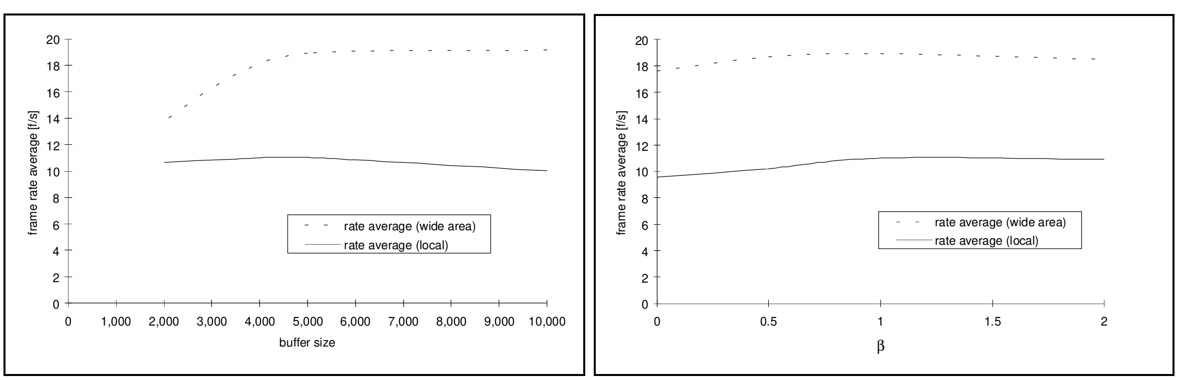

Figure 13: Frame rate average(configuration A) with 8 mux'ed sources and $\alpha = 3$; (a) for $\beta = 1$; (b) for a buffer size of 5,000 cells

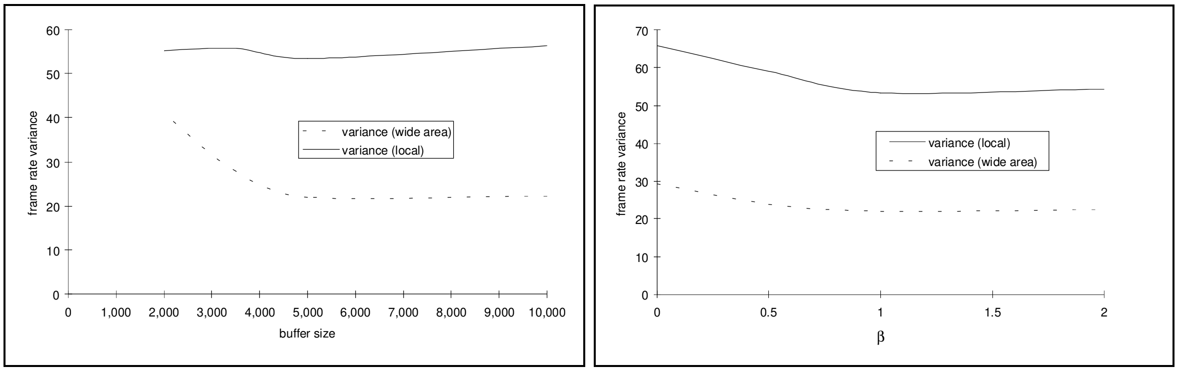

Figure 14: Variance of frame rate (configuration A) with 8 mux'ed sources and $\alpha=3$; (a) for $\beta=1$; (b) for a buffer size of 5,000 cells

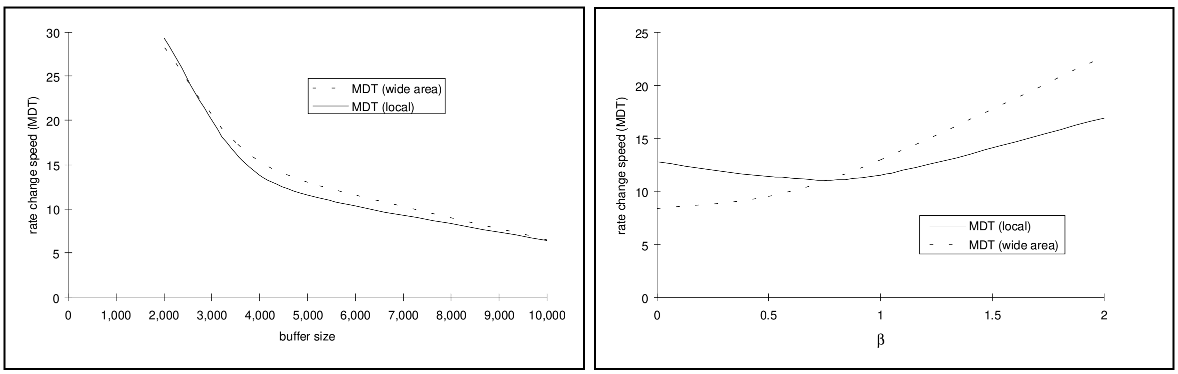

Figure 15: Mean rate change speed from frame to frame (MDT) with 8 mux'ed sources (configuration A) and $\alpha=3$; (a) for $\beta=1$; (b) for a buffer size of 5,000 cells

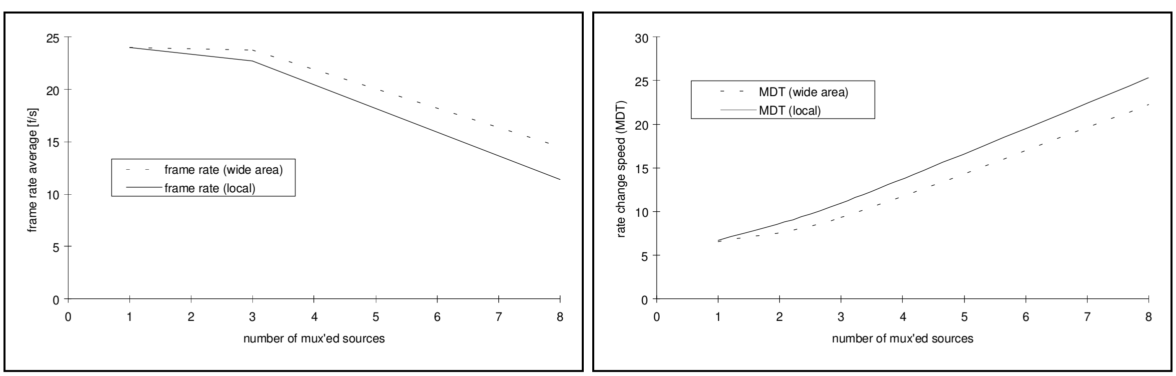

Figure 16: Dependency on number of mux'ed sources: (a) average frame rate, (b) mean rate change speed (MDT); configuration A, $\alpha=3$, $\beta=1$, buffer size of 2,000 cells

5.4.4 Frame Losses and Encoding Interrupts

Corresponding to the source behavior at high buffer occupancies defined in section 4.2.1 we distinguish between *frame losses* due to source buffer overflow and *encoding interrupts* (breaks) after the frame rate has been set to zero by the rate controller. The main difference is that in the second case the first frame after the break contains all changes of the screen content since the last frame transmitted correctly because the encoder knows about the interrupt.

As expected, losses and encoding interrupts (breaks) occur more frequent if the buffer size is smaller (see figure 17). At a buffer size of 5,000 cells the frame loss rate ranges around 10^{-4} and the ratio of break time to the total time is near 1%. This may be acceptable values for a low cost video transmission. But if the buffer size is smaller than 3,000 cells the number of losses and breaks grows rapidly. The reason for this are primarily very large frames rather than rate changes required by the network.

Figure 18 shows the influence of β. Again there is a local minimum at $\beta=1$. This seems to be a more general optimum for β, for it can be observed for each buffer size we investigated. The charts also show that the damping component (see section 4.2.3) causes a significant decrease of losses, at least for configuration A with 8 multiplexed sources. However, we experienced that the damping component should be omitted ($\beta=0$) if video sources are not multiplexed. If there are losses or interrupts at all, they occur more frequent if $\beta>0$.

The influence of the slope factor (α) is shown in figure 19. The larger α the stronger is the rate change function (RCF), and the more frames get lost. But this advantage is paid with more breaks (compare figure 19a with 19b). Nevertheless, as mentioned above breaks are more acceptable for video connections.

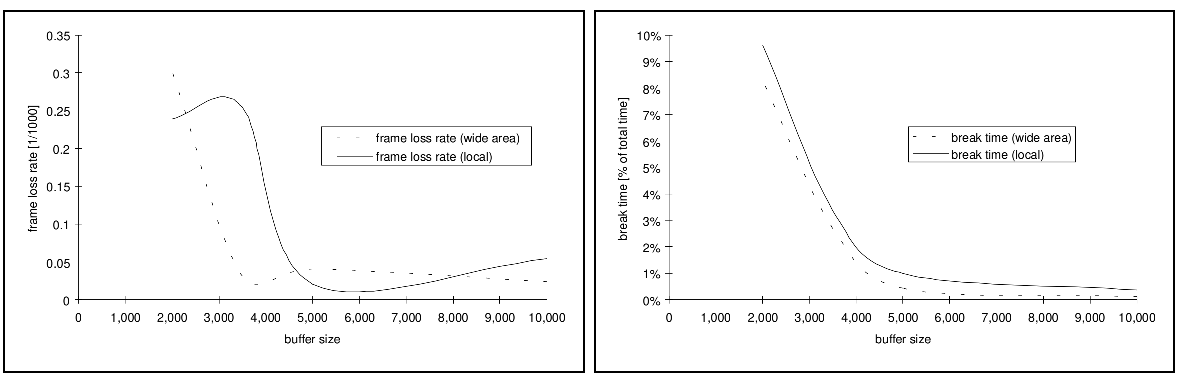

Figure 17: Frame loss rate (a) and break time (b) with 8 mux'ed sources (configuration A), $\alpha=3$ and $\beta=1$

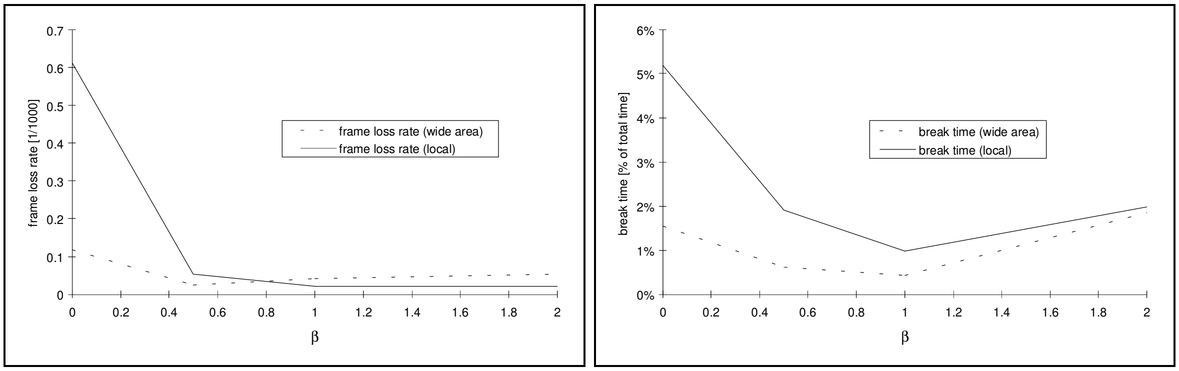

Figure 18: Frame loss rate (a) and break time (b) with 8 mux'ed sources (configuration A), α=3 and a buffer size of 5,000 cells

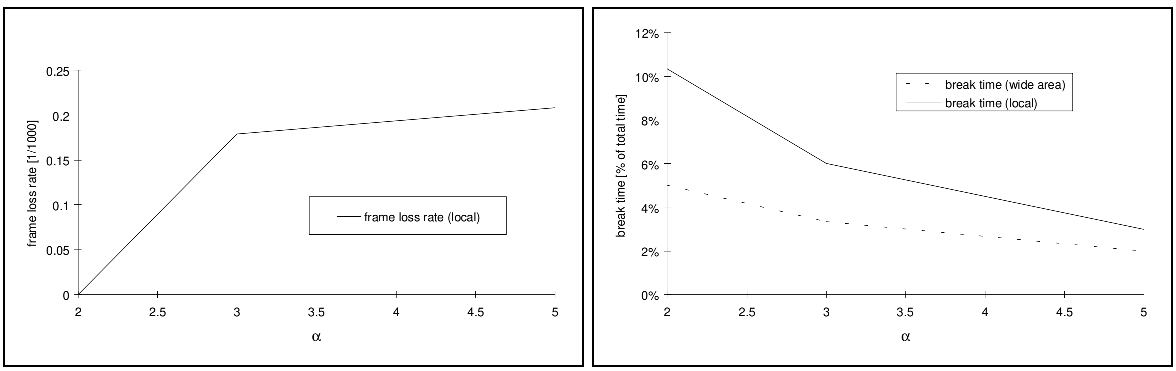

Figure 19: Dependency on the rate control slope factor (α): Frame loss rate (a) and break time (b) with 8 mux'ed sources (configuration A), β=1 and a buffer size of 3,500 cells. Note: The wide area source's buffer does not at all overflow here.

5.4.5 End-to-End Delay

The most important quality metric of an interactive video connection (e. g. videoconferences) is the *end-to-end delay*. We will evaluate this characteristic by the minimum required playout buffer given that the decoder shall be able to keep synchronization for *all* frames. Since we do not alter any source or link delay, the propagation delay (that must be added to the playout buffer) is nearly constant at 49 ms for the wide area source and 23 ms for the local source.

The larger the source buffer size the longer the cells are stored within the buffer, for the rate control tries to get $b=b_t$, and we made b is a function of the buffer size. Figure 20a quantifies this behavior. This figure shows that large buffers as they were used here are not applicable for interactive video, because the required playout buffer would be up to 1 second. But, even with a small buffer the playout puffer must be about 200 ms at minimum. Since this is the required value for the most delayed frame, the actual playout puffer is to be chosen even higher. Also, the propagation delay (a few ten milliseconds, see above) has to be added. The resulting end-to-end delay would be near 300 ms. This is hardly acceptable for interactive video. Therefore, smaller buffers are necessary. To achieve this while holding the losses bounded, we attempt giving up multiplexing. Figure 20b shows how this would decrease the required playout buffer. Note that, in our configuration, this includes the video sources do not demand the bandwidth. But, even if they would demand more than the fair share, the source buffer may be chosen smaller, say, 500 cells. Consequently, the end-to-end delay can achieve values those are acceptable for interactive video.

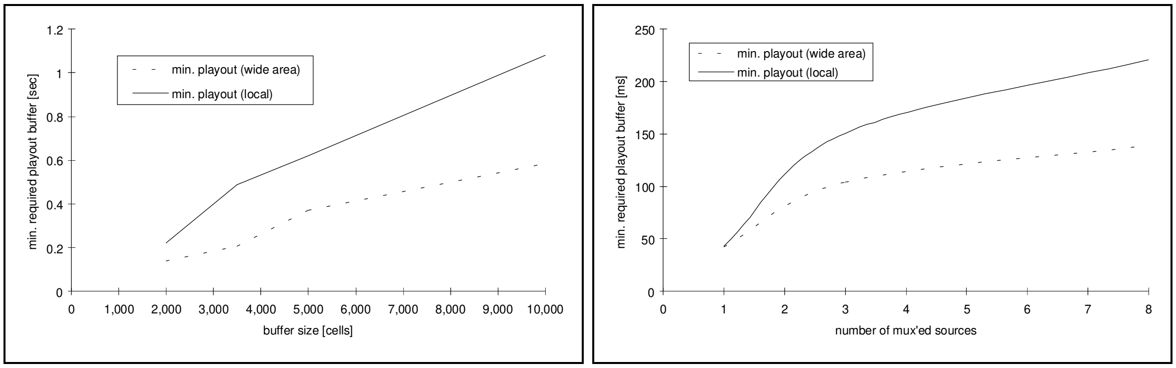

Figure 20: Minimum required playout buffer (configuration A, α=3, β=1): (a) as a function of the source buffer size (8 mux'ed sources); (b) as a function of the number of multiplexed sources (buffer size: 2,000 cells);

5.4.6 Fairness

Besides the video connection quality, a key issue of this study is the *fairness* among persistent and non-persistent sources in an ABR controlled network. If the bandwidth demand of all sources is higher than the fair share (applies to our configuration with 8 mux'ed sources), the ABR rate control should operate that all sources are sending at the fair share rate. But the share of the wide area source is much higher than that of the local source (figure 21). This is not a fair share but may be sufficient in most cases. Note that the share of the wide area source approaches the local source's share when the source buffer is small.

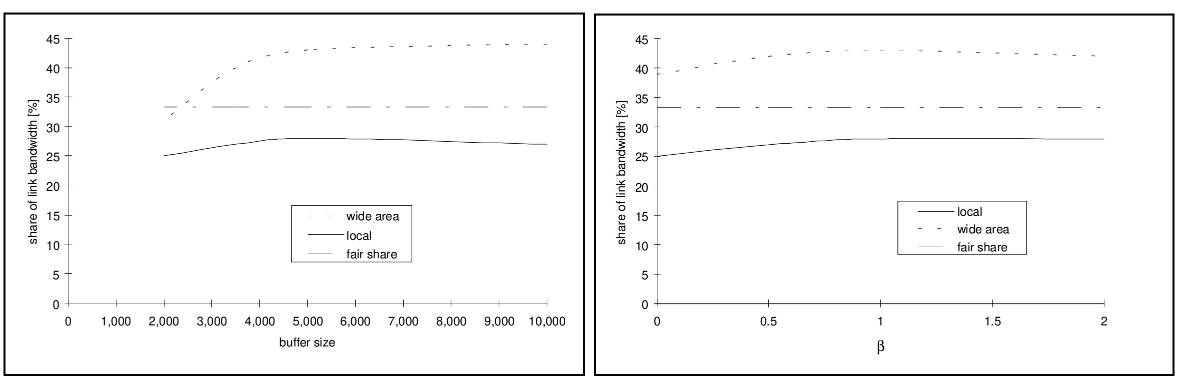

Figure 21: Fairness of video connections with configuration A (8 mux'ed sources) and α=3; (a) as a function of buffer size (β=1); (b) as a function of β (buffer size: 5,000 cells)

5.5 Acceptability of Frame Rate Changes for Human Users

So far we get only quantitative results which are appropriate to compare ABR video transmission using the proposed rate control with each other service suitable for video transmission. Finally, we now want to know whether the simulated quality of service is acceptable to human users. To get this we apply the simulated frame rate (see section 5.4.3) to a real video source. The original video used to build the trace driven source (see section 3.4) supplies this real source. Thus we can evaluate the result shown on screen subjectively. Since we have to exclude any influence of hardware performance, we employed the very fast video tool *nevit*[1]. We are aware that the evaluation should have been done by a large population. However, this would go beyond the limits of this study. Therefore, just the author and some colleagues have done this acceptability test.

As done before we assume a video conference transmission or a lecture distribution over a low cost connection. The ABR service is not feasible for high quality entertainment video. In this context we found the frame rate changes in a configuration with 8 multiplexed sources at a source buffer of 5,000 cells sufficient. Changes to low rates are perceptible but rare and thus not very disturbing. A larger buffer size does not bring an appreciable improvement. A smaller buffer size, however, makes visible rate changes much more frequent. Watching, e. g. a speaker's mouth is often impossible. This quality will be not acceptable for most people.

Note that these buffer sizes apply to the configuration assumed in section 5.4 (eight multiplexed sources supply a single buffer). But, since a real ABR source instance does not know how many video sources will ever be multiplexed at the regarded VC we had to assume a certain realistic value. If an ABR source is configured to carry only single video connections (strongly recommended) the buffer size may be set smaller respectively (≈500 cells). Thus, the end-to-end delay will become acceptable for interactive video as well.

---

[1] Available via `ftp://ftp.fokus.gmd.de/pub/step/nevit*`.

6 Summary and Future Work

In this study we evaluated whether the ABR service defined by the ATM-Forum is applicable for video transmission. We indeed did not want to create a new rate controlled ATM layer service which provides best properties for video transmission. Rather we investigated the video connection quality reachable while using the well-standardized ABR service. To get this, we designed a rate control mechanism which controls the video encoder using the network feedback provided by the ABR mechanism.

Our simulation results together with the acceptability test showed that the ABR service may be used for video transmission under the following conditions:

a) use of encoder rate control

b) large source buffer if many sources are multiplexed

c) no multiplexing if interactivity is required (small buffer)

d) rare visible quality loss is acceptable

According to these conditions ABR video is feasible only for scenarios as video teleconferencing or lecture distribution. This limitation will be acceptable, for ABR connections probably will be the low cost offers of future ATM providers. If interactivity is required (video conferencing) video sources should not been multiplexed in order to manage with small source buffer (delay).

An interesting result of our simulations is the difference between connection paths with different delays from the source to the next switch. Connections with longer delay provide better video quality, at least in our configuration. This phenomenon can be utilized by simply introducing additional delay. Nevertheless, because our “wide area” delay is only 8 msec, interactivity is preserved.

For minimizing the quality loss an effective rate adaptation system is necessary. The proposed approach may be improved by optimizing some of its parameters dynamically. Furthermore, a frame prediction scheme as proposed in [11] may be introduced, and the encoder control may be extended to support both frame rate and quantizer step size alteration.

Another aspect is the network configuration. The behavior of a network with much more than two or three VCs per link transmitting video should be investigated, too.

7 Bibliography

[1] Elwalid A., Heyman D., Lakshman T. V., Mitra D., Weiss A., “Fundamental Bounds and Approximations for ATM Multiplexers with Applications to Video Teleconferencing”, *IEEE Journal on Selected Areas in Communications: Special Issue on Fundamental Advances in Networking*, pp. 1004--1016, August 1995.

[2] Bernd Heinrichs and Markus Rossmann, “ABR: The ATM Service for Data Communication Applications?”, in First Workshop on ATM Traffic Management (WATM'95), (Paris, France), pp. 229--235, Dec. 1995.

[3] Daniel P. Heymann, Ali Tabatabei, and T. V. Lakshman, “Statistical analysis and simulation study of video teleconference traffic in ATM networks” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, pp. 49--59, Mar. 1992.

[4] D. P. Heyman and T. V. Lakshman, “Source Models for VBR Broadcast-Video Traffic,” in Proceedings of the Conference on Computer Communications (IEEE Infocom), (Toronto, Canada), pp. 664--671, June 1994.

[5] Raj Jain, Shivkumar Kalyanaraman, Sonia Fahmy, Rohit Goyal and Seong-Cheol Kim, “Tutorial Paper on ABR Source Behaviour”, ATM-Forum/96-1270, October 1996

[6] L. Kalampoukas, A. Varma, “Analysis of Source Policy in Rate-Controlled ATM Networks”, Computer Engineering Dept., University of California, Santa Cruz, 1996

[7] L. Kalampoukas, A. Varma, K. K. Ramakrishnan, “Dynamics of an Explicit Rate Allocation Algorithm for ATM Networks”, Computer Engineering Dept., University of California, Santa Cruz, 1996

[8] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, J. Jiang and S.-C. Kim, “Performance of TCP over ABR on ATM backbone and with various VBR traffic patterns”, ATM-Forum/96-1294, October 1996

[9] Hemant Kanakia, Partho P. Mishra, and Amy R. Reibman, “An Adaptive Congestion Control Scheme for Real Time Packet Video Transport,” *IEEE/ACM Transactions on Networking*, vol. 3, no. 6, pp. 671--682, Dec. 1995.

[10] Steve McCanne and Van Jacobson, “vic: A flexible framework for packet video,” in Proc. of ACM Multimedia '95, Nov. 1995.

[11] K. K. Ramakrishnan, T. V. Lakshman, and Partho P. Mishra, “Support for Compressed Video with Explicit Rate Congestion Control in ATM Networks”, in Proc. of The 6th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 96), Apr. 1996.

[12] G. Ramamurthy, Alex Kolarov, “A Source Traffic Model for Evaluation of ABR Service”, ATM Forum Traffic Management Working Group, ATM-Forum/94-0214, March 1994

[13] G. Ramamurthy, Alex Kolarov, Chinatsu Ikeda, Hiroshi Suzuki, “Evaluation of Rate Based Congestion Control Schemes for ABR Service - Wide Area Networks”, ATM Forum Traffic Management Working Group, ATM-Forum/94-0354, May 1994

[14] Amy R. Reibman and Arthur W. Berger, “Traffic Descriptors for VBR Video Teleconferencing Over ATM Networks,” *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 329--338, June 1995.

[15] A. R. Reibman and B. G. Haskell, “Constraints on Variable Bit-Rate Video for ATM Networks,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, pp. 361--372, Dec. 1992.

[16] A. Roberts, “Enhanced PRCA proportional rate control algorithm”, ATM Forum/94-0735R1, Aug. 1994.

[17] S. Sathaye, “ATM Traffic Management Specification Version 4.0”, ATM Forum/af-tm-0056.000, April 1996.

[18] Dorgham Sisalem, “Rate Based Congestion Control and its Effects on TCP over ATM”, Master's Thesis, Dept. of Telecommunications Engineering, Technical University of Berlin, May 1994.
Available from `ftp://ftp.fokus.gmd.de/pub/step/sa.da/Sisa9506:ATM.ps.gz`

[19] Dorgham Sisalem and Henning Schulzrinne, “Switch mechanisms for the ABR service: A comparison study,” in Telecommunication Distribution Parallelism (TDP'96), (La Londes Les Maures, France), Institut National des Telecommunications, Evry, June 1996.
Available from `ftp://ftp.fokus.gmd.de/pub/step/papers/Sisa9606:Switch.ps.gz`

[20] T. Turletti, C. Huitema, “RTP payload format for H.261 video streams”, RFC, IETF Audio-Video Transport Working Group, July 1995

[21] Brett J. Vickers and Tatsuya Suda, “An ATM service architecture for the transport of adaptively encoded live video”, in Proc. of ICCCN, Oct. 1996. Online available from `http://www.ics.uci.edu/~bvickers/papers/ic3n96/ic3n96.ps`

[22] L. Wojnaroski, “Baseline Text For Traffic Management Sub-Working Group”, ATM-Forum/94-0394R5, Oct. 1994

8 Appendix

8.1 List of Symbols

α	-	slope factor
β	-	damping factor
B	-	link bandwidth
b	-	buffer occupancy
$b_{\max}$	-	maximum buffer occupancy
b_s	-	buffer safety zone
b_t	-	target buffer occupancy
c	-	cell rate
f	-	frame rate
n	-	frame index
n_c	-	number of cells per frame
ρ	-	cell rate to frame rate ratio
T	-	total time
T_a	-	average interval
t_r	-	retry timer
Δt	-	frame duration

8.2 List of Abbreviations

ABR	-	Available Bit Rate
ACR	-	Allowed Cell Rate
ADTF	-	ACR Decrease Time Factor
BECN	-	Backward Explicit Congestion Notification
CBR	-	Constant Bit Rate
CCR	-	Current Cell Rate
CDF	-	Cutoff Decrease Factor
CI	-	Congestion Indication
CRM	-	Missing RM cell Count
EPRCA	-	Enhanced Proportional Rate Control Algorithm
ER	-	Explicit Rate
GCRA	-	Generic Cell Rate Algorithm
ICR	-	Initial Cell Rate
ISDN	-	Integrated Services Digital Network
MCR	-	Minimum Cell Rate
MDT	-	Mean Derivative by Time
NI	-	No Increase
Nrm	-	Maximum number of cells between RM cell generation
PCR	-	Peak Cell Rate
RCF	-	Rate Change Factor
RDF	-	Rate Decrease Factor
RIF	-	Rate Increase Factor
RM	-	Resource Management
TAT	-	Theoretical Arrival Time
TCR	-	Tagged Cell Rate
VBR	-	Variable Bit Rate
VC	-	Virtual Circuit

8.3 Class Definitions for the Trace Driven Source

```
// This file contains some useful class definitions to build a
// trace driven video source based on multiple traces.

// Author: Ronald Bollow, Dept. of Electrical Engineering,
// Telecommunication Networks Group, Berlin University of Technology

#ifndef _RB_VID_
#define _RB_VID_ 1

#define NUM 4                   // Number of traces

#include <pt_fstream.h>

class frame {
// This class represents a video frame to be read from a file.
public:
        frame(pt_ifstream *in) {
                *in >> number >> time >> cells;
                next = NULL;
        }
        frame() {
                number = 0;
                time = 0.0;
                cells = 0;
                next = NULL;
        }

        ~frame() { LOG_DEL; delete next; }

        int     getNumber () const{ return (number); }
        double  getTime () const{ return (time); }
        int     getCells () const{ return (cells); }
        frame*  getNext () const{ return (next); }

        void    setNext( frame* NextFrame ) { next = NextFrame; }



private:
        int number, cells;
        double time;
        frame* next;

};

class avg_frame {
// This class represents a video frame to be maintened depending on
// an averaging interval
public:
        avg_frame(double t, float fr, int Cells) {
                time = t;
                frameRate = fr;
                cells = Cells;
                next = NULL;
        }

        int isBefore(double t) {
                if (t > time) return 1; else return 0;
        }

        void setNext(avg_frame *NextFrame) { next = NextFrame; }

        int getCells() const { return (cells); }
        double getTime() const { return (time); }
        float getRate() const { return (frameRate); }
        avg_frame *getNext() const { return next; }
```

```
private:
        double time;
        float frameRate;
        int cells;
        avg_frame *next;
};

class VidSeq {
// This class represents video sequences described by NUM files each
// corresponding to a certain frame rate.
public:
        VidSeq(int *lenArray, int *frameRatesArray, char **fileNames) {
                int i;

                for (i=0; i<NUM; i++) {
                        frame *f;
                        int j;
                        pt_ifstream a_in;

                        // initialize
                        len[i] = lenArray[i];
                        frameRates[i] = frameRatesArray[i];
                        proc[i] = 1;
                        a_in.open(fileNames[i]);

                        // read in first frame
                        f = new frame(&a_in);
                        f_start[i] = f;
                        f_current[i] = f_start[i];

                        // read in remaining frames
                        for (j=1; j < len[i]; j++) {
                           f->setNext(new frame(&a_in));
                           f = f->getNext();
                        };
                        a_in.close();
                };
        }

        ~VidSeq() {
                int i;

                for (i=0; i<NUM; i++) {
                        LOG_DEL; delete f_start[i];
                };
        }

        void adjustPointers(double time, float fr) {
        // This function adjusts the pointers of both related sequences
        // in order to interpolate the next frame.
                int i;
                frame *f_new;

                frameRate = fr;
                setProc();
                for (i=0; i<NUM; i++) {
                   if (proc[i]) {
                        // set pointer to last frame before time
                        while (f_current[i]->getNumber()-0.5 <= (time * frameRates[i])) {
                           if ((f_new = f_current[i]->getNext()) == NULL) {
printf("\n%s\n\n","Error: No more frames read from file. -- Choose a higher value for \"len\"!
...exiting");
                                exit(1);
                           };
                           f_current[i] = f_new;
                        };
                        if ((f_new = f_current[i]->getNext()) == NULL) {
printf("\n%s\n\n","Error: No more frames read from file. -- Choose a higher value for \"len\"!
...exiting");
                           exit(1);
                        };
                        // if time is more close to next frame: set appropriate flag
                        if ( (time * frameRates[i] - f_current[i]->getNumber()) >
```

```
                          (f_new->getNumber() - time * frameRates[i]) )
                             flag[i] = 1;
                        else flag[i] = 0;
                    };
                }
        }

        frame   **getStart () { return (f_start); }
        frame   **getCurrent () { return (f_current); }
        float   getFrameRate () const{ return (frameRate); }

        int     getCells () {
        // This function computes the interpolated number of cells from the frames
        // choosen by adjustPointers() (must have been called before!)

                int i, j;
                int cells[2];
                float fr[2];

                j=0;
                for(i=0; i<NUM; i++) {
                        if(proc[i]) {
                                fr[j] = frameRates[i];
                                if(flag[i]) cells[j++] = f_current[i]->getNext()->getCells();
                                else cells[j++] = f_current[i]->getCells();
                        };
                }
                return((int)((frameRate-fr[0])*(cells[1]-cells[0])/(fr[1]-fr[0]))+cells[0]);
        }

private:
        void setProc() {
        // This function sets the proc array which is used to process the (two)
        // necessary sequences only.

                if (frameRate <= frameRates[1]) {
                        proc[0] = 1;
                        proc[1] = 1;
                        proc[2] = 0;
                        proc[3] = 0;
                }
                else if (frameRate <= frameRates[2]) {
                        proc[0] = 0;
                        proc[1] = 1;
                        proc[2] = 1;
                        proc[3] = 0;
                }
                else {
                        proc[0] = 0;
                        proc[1] = 0;
                        proc[2] = 1;
                        proc[3] = 1;
                };
        }

        int len[NUM];           // Number of frames for each trace
        int proc[NUM];          // TRUE if trace must be processed
        int flag[NUM];          // TRUE if time*frameRates[x] is closer
                                // to f_current[x]->getNext()->getTime()
        int frameRates[NUM];    // frame rate for each trace
        float frameRate;        // current frame rate
        frame *f_start[NUM];    // Pointers to first frames
        frame *f_current[NUM];  // Pointers to current frames -- set by adjustPointers()

};

#endif
```

8.4 Source Code of the Controlled Video Source

We used the Ptolemy simulation system. User defined parts are added to this system as so called *stars*. Here is the source code of the star performing the trace driven video source and the rate control.

```
defstar{
        name { VarSyncSrc }
        domain { DE }
        author { Ronald Bollow }
        desc {
This star simulates a trace driven video source adapting its output rate
(by varying the frame rate) to the available transmission rate provided by the
network. For this purpose it gets information about either the allowed cell rate
(ACR) or the ABR source buffer occupancy.
        }

        input   {
                name { ACR_in }
                type { float }
                desc { The allowed cell rate provided by the network. }
        }
        input   {
                name { Queue }
                type { int }
                desc { The actual source queue occupancy. }
        }
        input   {
                name { Next }
                type { float }
                desc { Trigger input from Demand }
        }
        output {
                name { Output }
                type { float }
                desc { Output of data cells (as float particles containing frame index) }
        }
        output {
                name { Demand }
                type { float }
                desc { Delay until sending the next cell }
        }

        defstate {
                name { Use_Queue }
                type { int }
                default { 1 }
                desc { if 1 then use of buffer occupancy is switched on }
        }
        defstate {
                name { PCR }
                type { float }
                default { "Rate" }
                desc { Peak Cell Rate }
        }
        defstate {
                name { ICR }
                type { float }
                default { "Rate/20" }
                desc { The initial cell rate (ACR based only) }
        }
        defstate {
                name { IFR }
                type { float }
                default { 6 }
                desc { The initial frame rate (buffer based only }
                }
        defstate {
```

```
		name { Interval }
		type { float }
		default { 1.0 }
		desc { The rate averaging interval (ACR based only) }
	}
	defstate {
		name { Down_Pressure }
		type { float }
		default { 1.0 }
		desc { The ratio of the target cell rate to the ACR. }
	}
	defstate {
		name { Max_Buffer }
		type { int }
		default { 9500 }
		desc { That source buffer occupancy for which rate is set to zero. }
	}
	defstate {
		name { Target_Buffer }
		type { int }
		default { 4750 }
		desc { The target source buffer occupancy. }
	}
	defstate {
		name { Scale_Factor }
		type { float }
		default { 3.0 }
		desc { for computing new rate from buffer occupancy }
	}
	defstate {
		name { Damp_Factor }
		type { float }
		default { 1.0 }
		desc { for computing new rate from buffer occupancy }
	}
	defstate {
		name { Frame_Rates }
		type { intArray }
		default { "3 6 12 24" }
		desc { The frame rates of the Traces in files (1..4). }
	}
	defstate {
		name { Source_File_1 }
		type { string }
		default { ""}
		desc { File that specifies source with frame rate #1 (lowest rate). }
	}
	defstate {
		name { Source_File_2 }
		type { string }
		default { ""}
		desc { File that specifies source with frame rate #2. }
	}
	defstate {
		name { Source_File_3 }
		type { string }
		default { ""}
		desc { File that specifies source with frame rate #3. }
	}
	defstate {
		name { Source_File_4 }
		type { string }
		default { ""}
		desc { File that specifies source with frame rate #4 (highest rate). }
	}
	defstate {
		name { Lengths }
		type { intArray }
		default { "10000 20000 40000 80000" }
		desc { The numbers of frames to be read from each Source_File. }
	}
	defstate {
		name { Delay }
```

```
        type { float }
        default { "0.0" }
        desc { Time offset pointing into the trace. }
}
defstate {
        name { First_Frame }
        type { int }
        default { 1 }
        desc { start of frame number counting }
}
defstate {
        name { Debug_File }
        type { string }
        default { "{vidsrc.log}"}
        desc { The log file. }
}

hinclude { "pt_fstream.h","vidseq.h" }

protected {
        pt_ofstream *p_out;
        char *FileNames[4];
        float CCR, ACR, fr, B;
        double frame_start, next_time, interval;
        int i, fi, ci, cn, q, q_last, cell_sum, proc_q;
        avg_frame *f, *f2;
        VidSeq *t;
}

setup{
        t = NULL;       // pointer to the video sequence
}

begin {
        LOG_DEL; delete p_out;
        LOG_NEW; p_out = new pt_ofstream(Debug_File);

        FileNames[0] = Source_File_1;
        FileNames[1] = Source_File_2;
        FileNames[2] = Source_File_3;
        FileNames[3] = Source_File_4;

        LOG_DEL; delete t;
        LOG_NEW; t = new VidSeq(Lengths, Frame_Rates, FileNames);

        f = NULL;             // pointer to oldest average frame
        fr = IFR;             // current frame rate
        ACR = ICR;            // initially assumed ACR
        CCR = ACR;            // current cell rate
        fi = First_Frame - 1;   // frame index
        ci = 0;               // cell index per frame
        cn = 0;               // number of cells per frame
        cell_sum = 0;         // sum of cells in average interval
        B = float(Scale_Factor);
        q = 0;                // current buffer occupancy
        q_last = 0;           // buffer occupancy at last frame
        proc_q = 1;           // process input port "Queue"
}

constructor {
        p_out = 0;
        ACR_in.triggers();
        Queue.triggers();
        ACR_in.before(Queue);
        Queue.before(Next);
}
destructor {
        LOG_DEL; delete p_out;
        p_out = 0;
}

wrapup {
        LOG_DEL; delete p_out;
```

```
              p_out = 0;
               LOG_DEL; delete t;
        }

        go {
           completionTime = arrivalTime;
          pt_ofstream& OUTPUT = *p_out;

           if (ACR_in.dataNew)
           { // a new ACR value is received
                ACR = ACR_in.get();

                if (!Use_Queue) CCR = float(Down_Pressure) * ACR;
           }

          if (Queue.dataNew)
          { // a new buffer occupancy value is received

              Queue.dataNew = FALSE;
              if (Use_Queue && proc_q) {
                     int Dbuf;

                     q_last = q;
                     q = Queue.get();
                     Dbuf = q - Target_Buffer;
                     proc_q = 0;

                     // compute new frame rate
                     if (q > Max_Buffer) fr = 0;
                     else {
                        // Rate adaptation function
                        fr = fr * B/(B-1) * (1 + 1/((B-1) * Dbuf / (Max_Buffer - Target_Buffer)
- B) + ((q_last-q)*float(Damp_Factor)/float(Max_Buffer)) );

                         if (fr > Frame_Rates[3]) fr = Frame_Rates[3];   // max. frame rate
                        else if ((fr < Frame_Rates[0]) && (fr > 0)) fr = Frame_Rates[0];

                        if (fr < 0) fr = 0;
                     }

              };
          }

           if (Next.dataNew)
           { // The star is refired by itself (Demand output)

               Next.dataNew=FALSE;
               if (!ci++) {    // first cell of frame
                     if (fr <= 0) {
                            ci = 0;
                            fr = 1.0;
                            proc_q = 1;
                             Demand.put(completionTime) << 1.0;
                            return;
                     };
                     if (Use_Queue) {
                              // read out next frame from trace
                              t->adjustPointers(completionTime+double(Delay), fr);
                              cn = t->getCells();
OUTPUT<< "Q " << q << " fr " << fr << " cn " << cn << " fi " << (fi+1) << " at " << completionTime
<<"\n";
                     } else {
                          if (f!=NULL) {
                               int i;
                               float fr_sum;

                               f2 = f;
                               // delete avg frames that are too old
                               while (f->isBefore(completionTime-(double)Interval)) {
                                       f2 = f->getNext();
                                       LOG_DEL; delete f;
                                       f = f2;
                                    if (f==NULL) break;
```

```
                              }
                            if (f==NULL) fr = CCR / 130;
                            else {
                                 cell_sum=0; fr_sum=0;
                                 i=1;
                                 // averaged cells to frame rate ratio
                                 while(f2->getNext()!=NULL) {
                                         cell_sum += f2->getCells();
                                         fr_sum += f2->getRate();
                                         f2 = f2->getNext();
                                         i++;
                                 }
                                 // f2 now points to last avg frame
                                 cell_sum += f2->getCells();
                                 fr_sum += f2->getRate();
                               // determine the actual interval (<=Interval)
                                 interval = f2->getTime() - f->getTime() + double(1/f2-
>getRate());
                                 // compute the new frame rate
                                 fr = CCR * (fr_sum/i) / (cell_sum/interval);
                            }
                              if (fr > Frame_Rates[3]) fr = Frame_Rates[3];      // max. frame
rate
                            else if ((fr < Frame_Rates[0]) && (fr > 0)) fr = Frame_Rates[0];

                              // read out next frame from trace
                              t->adjustPointers(completionTime+double(Delay), fr);
                              cn = t->getCells();
                            if(f==NULL) { // if  all avg frames were deleted
                                   LOG_NEW; f = new avg_frame(completionTime, fr, cn);
                            } else {// if there are still avg frames
                                   // append the new one to avg frames
                                   LOG_NEW; f2->setNext(new avg_frame(completionTime, fr, cn));
                            }
OUTPUT<< "fr " << fr << " cn " << cn << " frame_no " << (fi+1) << " at " << completionTime <<"\n";
                         }
                         else {          // if no avg frame available
                              fr = CCR / 130;      // 130: assumed mean cells per frame value
                              if (fr > Frame_Rates[3]) fr = Frame_Rates[3];
                            else if ((fr < Frame_Rates[0]) && (fr > 0)) fr = Frame_Rates[0];
                              t->adjustPointers(completionTime+double(Delay), fr);
                              cn = t->getCells();
OUTPUT<< "fr " << fr << " cn " << cn << " frame_no " << (fi+1) << " at " << completionTime <<"\n";
                              LOG_NEW; f = new avg_frame(completionTime, fr, cn);
                         };
                    };
                      fi++;
                      frame_start = completionTime;
                };

                if (ci==cn) {   // last cell of frame
                        double delay;

                        ci=0;   // reset cell index
                      proc_q = 1;
                        // compute delay until next firing (next frame)
                        next_time = 1 / t->getFrameRate()
                                  + frame_start
                                  - completionTime;

                        // report if delay is too small or negative
                        if ((delay = (1/double(PCR) - next_time)) >= 0.0 ) {
printf("%s%d%s%f%s%f\n","fi ",fi," delay ",delay," at ",completionTime);
                                next_time = 1/(double)PCR;
                        };
                }

                else {  // for all cells of frame except the last in frame
                        // compute delay until next firing (same frame)
                        next_time = 1/(double)PCR;
                };

                // put the delay to Demand output
```

```
                Demand.put(completionTime) << (float)next_time;

                // put the event causing a cell creation to Output
                Output.put(completionTime) << (float)fi;
           }

        }
}
```