Multimedia Systems 7: 70–90 (1999)

Multimedia System

Synchronized delivery and playout of distributed stored multimedia streams

Ernst Biersack, Werner Geyer

Institut Eurécom, 2229 Route des Crêtes, F-06904 Sophia-Antipolis, France; e-mail: erbi@eurecom.fr

Abstract. Multimedia streams such as audio and video impose tight temporal constraints for their presentation. Often, related multimedia streams, such as audio and video, must be presented in a synchronized way. We introduce a novel scheme to ensure the continuous and synchronous delivery of *distributed stored* multimedia streams across a communications network. We propose a new protocol for synchronized playback and compute the buffer required to achieve both, the continuity within a single substream and the synchronization between related substreams. The scheme is very general and does not require synchronized clocks. Using a resynchronization protocol based on buffer level control, the scheme is able to cope with server drop-outs and clock drift. The synchronization scheme has been implemented and the paper concludes with our experimental results.

Key words: Sychronization – Multimedia streams – Video server

1 Introduction

1.1 Motivation

Advances in communication technology lead to new applications in the domain of multimedia. Emerging high-speed, fiber-optic networks make it feasible to provide multimedia services such as video on-demand, tele-shopping or distance learning. These applications typically integrate different types of media such as audio, video, text or images. Customers of such a service retrieve the digitally stored media from a *video server* [Ber96] for playback.

1.2 Multimedia synchronization

Multimedia refers to the integration of different types of data streams, including both *continuous media* streams (audio and video) and *discrete media* streams (text, data, images). A certain temporal relationship exists between the information units of these streams. Multimedia systems must mai this relationship when storing, transmitting and prese the data. Commonly, the process of maintaining the tem order of one or several media streams is called *multim synchronization* [Eff93].

One can distinguish between *live synchronization* for media streams and *synthetic synchronization* for stored dia streams [Ste95]. In the former case, the capturing playback must be performed almost at the same time, w in the latter case, samples are recorded, stored and pl back at a later time. For live synchronization, e.g., in conferencing, the tolerable end-to-end delay is in the der of a few hundred milliseconds only. Consequently, size of an elastic buffer at the receiver must be kept sr trading-off requirements for jitter compensation against delay for interactive applications. Synthetic synchroniza of recorded media stream is easier to achieve than live chronization: higher end-to-end delays are tolerable, and fact that sources can be influenced proves to be very ad tageous as will be shown later. It is, for instance, poss to adjust playback speed or to schedule the start-up ti of streams as needed. However, as resources are limite is desirable for both kinds of synchronization to keep required buffers as small as possible [Koe94].

Continuous media are characterized by a well-defi temporal relationship between subsequent data units. In mation is only conveyed when media quanta are prese continuously in time. As for video/audio, the temporal r tionship is dictated by the sampling rate. The problem maintaining continuity within a single stream is refe to as *intra-stream* synchronization or serial synchron tion [Bul91]. Moreover, there exist temporal relationsl between media units of related streams, for instance, an dio and video stream. The preservation of these temp constraints is called *inter-stream* synchronization. To sc the problem of stream synchronization, we must regard b issues which are tightly coupled.

Correspondence to: E. Biersack

1.3 Intra- and inter-stream synchronization

A continuous media stream consists of a sequence of (encoded)[1] samples which are transferred between source and sink. The task of intra-stream synchronization is to maintain the inherent temporal properties given by the sampling rate, i.e., information has to be reproduced as originally captured[2] [Scr92]. The temporal relationship within a single stream is mainly disturbed for the following reasons [Blu94; Lit92; Scr92]:

- *network jitter,*
- *end-system jitter,*
- *clock drift,*
- *changing network conditions.*

Network jitter denotes the varying delay that stream packets experience on their way from the sender to the receiver network I/O device. It is introduced by buffering in intermediate nodes. *End-system jitter* refers to the variable delays arising within the end-systems, and is caused by varying system load and the packetizing and depacketizing of media units with variable size, which are passed through the different protocol layers. Jitter is commonly equalized by the use of an elastic buffer at the sink [Blu94].

Capturing, reproduction and presentation of continuous media is driven by end-system clocks. In general, clocks cannot be assumed to be synchronized. Due to temperature differences or imperfections in the crystal clock, the frequency of end-system clocks can differ over a long period of time. The result is an offset in frequency to real time and to other clocks, which causes a drift rate from 10^{-6} s/s up to 10^{-3} s/s.[3] The problem of clock drift can be coped with by using time-synchronizing protocols within a network. The network time protocol (NTP), for instance, offers a global (virtual) time to its service users [Mil91]. Otherwise, if the problem of clock drift is neglected, buffer overflow or buffer starvation at the client will arise over a long period of time [Scr92; Scr95]. The effect of clock drift is also known as *skew*, which is defined as an average jitter over a time interval [Lit92].

Changing network conditions, not introduced by jitter, refer to a variation of connection properties, for instance, an alteration of the average delay or an increasing rate of lost media units.[4] These effects strongly depend on the QoS the underlying network can provide. However, synchronization mechanisms need to cope with this kind of problem.

[1] Commonly used digital compression techniques are, for instance, JPEG, MPEG, or H.261.

[2] There exist exeptions, of course, when, for example, the playback rate is altered to achieve VCR functions like fast-forward.

[3] A drift of 10^{-6} s/s is a more common value for today's systems. If the problem of synchronization is restricted to the playback of a single video, for instance, with a duration of 90 min, a total asynchronity of 5.4 ms will arise. This deviation cannot be perceived by a user. On the other hand, a drift of 10^{-3} s/s resulting in an asynchrony of 5.4 s will strongly influence presentation quality [Scr92].

[4] Commonly, unconfirmed datagram services are used for transmitting multimedia data. Retransmissions are not suitable as data is extremely time critical. A datagram service is unreliable and media units are lost from time to time. The proposed sychronization scheme handles lost media units such that the last media unit is displayed again, that is, the last media unit is doubled.

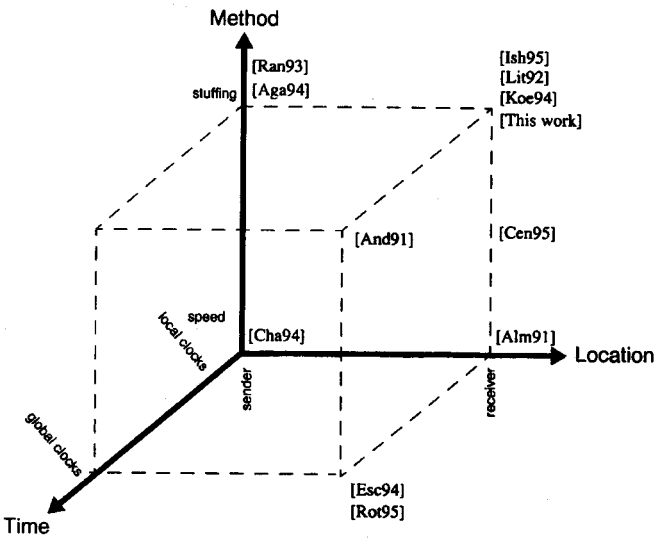

Fig. 1. Classification of synchronization mechanisms [Koe94]

Inter-stream synchronization must establish and maintain a certain temporal relationship between two or more related continuous media streams. Little et al. [Lit91] present 13 possible relationships between time intervals of related streams. Common relations are interval *a equals b* or *a before b*. Inter-stream synchronization must ensure these relations to a certain accuracy determined by the end user.

1.4 Related work

A common classification scheme for synchronization approaches does not exist. At present, synchronization mechanisms are either application-specific or try to cover synchronization on a more abstract level independent of the application at hand. Surveys of multimedia synchronization mechanisms can be found in [Ehl94], [Koe94] and [PL96]. The January 1996 issue of the Journal of Selected Areas in Communications is also devoted to synchronization [Geo96]. In order to give a rough overview of existing approaches, we adopted a classification scheme of Koehler et al. [Koe94]. We are using the following three important classification criteria[5] to systematize existing work in a *3D cube* (see Fig. 1).

- *Time.* Schemes can be classified whether or not they assume a synchronized, global time within a network. *Global clocks* allow to compensate for clock drift and to calculate exact values for delay or jitter. On the other hand, they are not available on every system at present, and accuracy is not always as fine-grained as desired. A sophisticated, complex protocol is required.
- *Location.* Synchronization functionality may be located either at the source or at the sink, depending on the capabilities of the sink [Koe94].
- *Method.* Restoring synchronization can be done either by speeding up or slowing down presentation or production of media units, or by *stuffing*. The latter can be performed either by duplicating/deleting media units

[5] Further criteria are, for instance, *participants*, *direction*, and *flexibility* (see [Gey95]).

or by pausing/skipping, respectively. The effect of the methods is the same. By applying, a stream that is out of synchronization may either catch up or slow down in order to regain synchronization. Changing speed might not always be possible as resources, e.g., bandwidth, are limited [Koe94; Ste95].

The work of Steinmetz [Ste90] and Little et al. [Lit90] is not classified in the cube because they do not present a concrete synchronization scheme, but examine synchronization issues in a more abstract way. Steinmetz discusses characteristics of multimedia systems and presents a set of constructs to express inter-media relationships. Little et al. model the inter-media timing based on timed Petri nets.

Escobar et al. [Esc94] and Rothermel et al. [Rot95] propose a scheme that requires globally synchronized clocks. Their synchronization mechanism relies on time stamps to determine the different kind of delays each stream experiences. At the receiver, different delays are equalized to the maximum delay by buffering. Rothermel et al. enhance this basic mechanism with a *buffer-level control* and a *master-slave* concept. The usage of a logical time system (LTS) proposed by Anderson et al. [And91] is very similar to global clocks.

Rangan et al. [Ran93] present a synchronization technique based on feedback. Synchronization is done at the sender side, assuming that the receiver stations send back the number of the currently displayed media unit. Asynchrony can be discovered by the use of so-called relative time stamps (RTS). Synchrony is restored by deleting or duplicating media units. Trigger packets are exchanged periodically so as to calculate the relative time deviation between sender and receiver. Agarval et al. [Aga94] adopt the idea of Rangan et al. and enhance the scheme by dropping the assumption of bounded jitter.

Chakrabiti et al. [Cha94] also apply feedback: the sender's production rate is controlled by the used buffer space at the receiver. A receiver clock determines a constant consumption rate from the buffer.

The technique of phase-locked loops is usually applied to restore synchronization of continuous data transmitted via an asynchronous network, e.g., ATM. The buffer level at the receiver is compared to a nominal value. Basically, the readout clock is driven by the fill level. A phase-locked loop mechanism is described by Almeida et al. [Alm91].

A more pragmatic solution for synchronization is given by Cen et al. [Cen95]. It is based on software feedbacks to the sender. Synchronization is regained either by influencing production speed or by skipping/pausing. To filter out jitter effects, low-pass filters are used.

Little et al.'s skew control system [Lit92] applies a kind of buffer level control. Within a certain nominal buffer level inter-stream synchronization is maintained. When defined thresholds are reached, synchronization is regained by duplicating or dropping media units. Little assumes a constant playout rate and guaranteed network resources. The synchronization scheme of Koehler et al. [Koe94] covers intra-stream synchronization exclusively. The mechanism is based on controlling the receiver buffer level. The fill level is filtered, compared to a nominal value, and evaluated by a control function. Correspondingly, intra-stream synchronization is restored by duplicating or deleting media units in the lo buffer.

We propose a synchronization scheme for stored m dia that achieves both, suitable intra- and inter-stream sy chronization. The scheme is receiver-based and does r assume global clocks. Resynchronization is done by sk ping/pausing (see Fig. 1). Our work has been inspir by the ideas of Ishibashi et al. [Ish95], who propose achieve inter-stream synchronization by providing int stream synchronization for each stream involved, and Santoso [San93] who provides conditions for a smooth pl out. Ishibashi et al. propose a time-stamp-based synch nization and apply a concept based on delay estimatic to perform synchronization in case of unknown delay. On intra-stream synchronization is established, inter-stream sy chronization can be maintained with a certain probabili To initiate the playback of a stream in a synchronized ma ner, we introduce a novel *start-up protocol*. For resynch nization, we present a buffer level control concept, whi has been influenced by Koehler et al. and Rothermel et [Koe94; Rot95].

1.5 Context of the synchronization problem

The synchronization problem addressed in this paper is m tivated by our work on scalable video servers. We have c signed and implemented a video server, called *server c ray*, consisting of n *server nodes* (see Fig. 2). A video distributed over all server nodes using a technique call *subframe striping*. Each video frame f_i is partitioned ir n equal-size parts $c_{i,j}$, called *subframes*, which are stor on the n different servers. If $F_i = \{c_{i,1}, \ldots, c_{i,n}\}$ deno the set of subframes for f_i, then $f_i = \bigcup_{j=1\ldots n} c_{i,j}$. The serv array with the synchronization mechanisms presented in tl paper has been successfully implemented in our video serv prototype [Ber96].

During playback, each server node is continuously tra mitting its subframes to the client. The transfer is schedul such that all striping blocks that are part of the same fra are completely received by the client at the deadline of t corresponding frame. The client reassembles the frame combining the subframes from all server nodes. An exa ple for $n = 3$ with each server sending with a rate of subframes per second is depicted in Fig. 3.

Another example for inter-stream synchronization stored multimedia streams is given by Cen et al. [Cen9 They describe a distributed MPEG player with the auc server and the video server being at different locations the Internet environment.

2 Synchronization protocol

2.1 Overview

We derive our synchronization scheme by stepwise refir ment. First, we develop a solution for the case of zero jitt and then relax this assumption, requiring bounded jitter on Finally, we cover synchronization problems not introduc

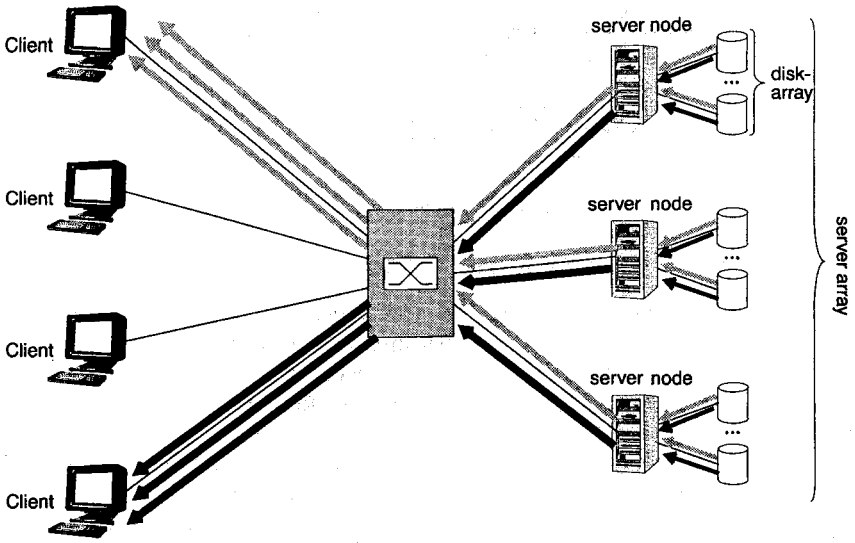

Fig. 2. Server array

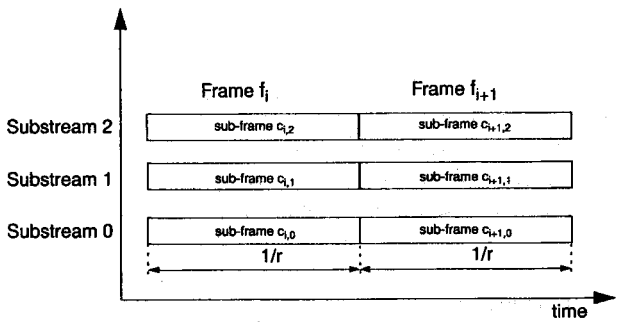

Fig. 3. Temporal relationship for subframe striping

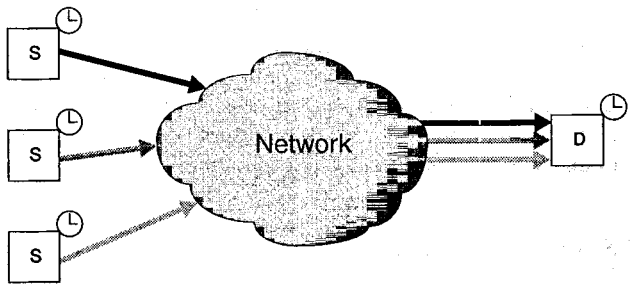

Fig. 4. Distributed architecture for the synchronization scheme

by jitter. In each step, we derive the buffer requirements and playout deadlines to assure inter- and intra-stream synchronization. We present three models.

Model 1 solves the problem of *different but fixed delays* on the network connections for each substream. We propose a novel synchronization protocol that compensates for these delays by computing well-defined starting times for each server. The protocol allows to initiate the synchronized playback of a media stream that is composed of several substreams.

Model 2 takes into account the *jitter* experienced by media units traveling from the source to the destination. Jitter is assumed to be bounded. To smooth out jitter, elastic buffers are required. Our scheme guarantees a smooth playback of the stream and has very low buffer requirements. Model 2 covers both intra-stream synchronization and inter-stream synchronization.

Model 3 solves the problems of *clock drift*, *changing network conditions*[6] and *server drop outs* by employing a buffer level control with a feedback loop to the servers so as to regain synchronization in the case of disturbances. Again, buffer requirements are regarded with respect to the results of models 1 and 2. The behavior of a filtering function is examined. Filters are necessary to identify whether a problem is of long-term or short-term effect. The tuning of some parameters is discussed.

[6] Because synchronization is strongly affected by an alteration of the average delay, we focus on this issue.

Novel aspects of our work are:

- the start-up protocol and the concept of shifting the starting times,
- a stepwise approach that gradually relaxes the assumptions and therefore leads to a whole family of synchronization protocols,
- a rigorous derivation (with proofs) for the exact buffer consumption,
- a working implementation of the proposed scheme that addresses various practical aspects such as the choice of the parameters and the resynchronization strategy, and allows us to obtain experimental results.

For the proposed synchronization scheme, we assume that a client D is receiving substreams from different servers.[7] Client and servers are interconnected via a network (see Fig. 4).

Each of the servers denoted by S delivers an independent *substream* of *media units*.[8] The production rate is driven by the server clock. Arriving media units are buffered in FIFO queues at the destination D. The playout of the entire *stream*,

[7] It is also possible that a *single* server sends *multiple* substreams to a client. Our model is more general and covers this case too.

[8] We use the term *media unit* in a broader sense. Depending on the strategy of striping, a media unit can be a complete frame or only a subframe (see Sect. 1.5). *Inter-frame* striping distributes complete frames across all server nodes whereas *subframe* striping divides a frame into n equal parts called subframes which are then distributed across server nodes. The presented model is general enough to handle both strategies. In the paper, we always refer to subframe striping in order to ease understanding.

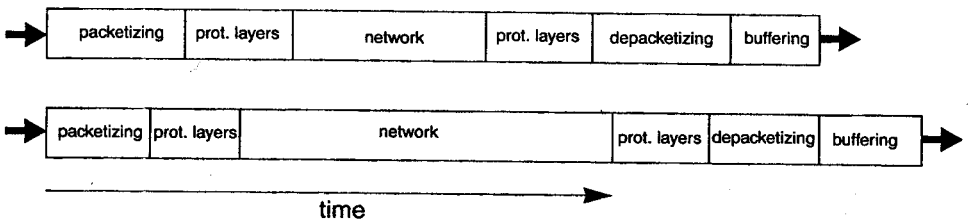

Fig. 5. Different delays experienced by independent substreams

composed of all the substreams, is driven by the destination's clock.

While the use of globally synchronized clocks facilitates synchronization, our synchronization scheme does *not* assume the presence of a global time or synchronized clocks for several reasons [Ran93].

- Network time protocols are not ubiquitous.
- Clock synchronization needs sophisticated, complex protocols.
- Heterogenous subnetworks hinder the presence of a global time because different organizational domains do not wish to synchronize their time with other domains.
- Even when using synchronized clocks, errors are introduced by inaccuracies and, thus, mechanisms are needed in any case to cope with this problem.

2.2 Sources of asynchrony

Several sources of asynchrony exist in the configuration described in the previous section. These are *different delays*, *network jitter*, *end-system jitter*, *clock drift*, *alteration of the average delay*, and *server drop-outs*. Specifically the assumption of independent network connections imposes different delays. A synchronization scheme has to compensate for these differences in order to display the continuous media stream in a timely order. Beside the network delay, media units experience a delay due to *packetizing/depacketizing*, the processing through the lower protocol layers, and the buffering at the client. Figure 5 shows the cumulative delay of two independent substreams which are started at the same time. Due to different delays, the media units of the two substreams are not played out simultaneously. The variation of delay is defined as *jitter*. Furthermore, the synchronization scheme has to be adaptive with respect to a change of the average delay (*changing network conditions*) and to server drop-outs, which are a realistic assumption when using non-real-time operating systems.

2.3 Assumptions

Our synchronization mechanism uses *time stamps*. Each time a media unit[9] is scheduled by a server, it is stamped with the current local time in order to allow the client to calculate the roundtrip delay, jitter, or inter-arrival times. Moreover, we assume that each media unit carries a *sequence number* for determining the media unit order.

In contrast to other approaches, buffer requirements or fill levels are always stated in terms of media units or time, instead of the amount of allocated memory. This consideration is preferred because synchronization is a problem of time and, for continuous media, time is represented implicitly by the media units of a stream. This seems reasonable because media unit sizes vary due to encoding algorithms like JPEG or MPEG [Koe94]. However, notice that a mapping of media units to the allocation of bytes must be carried out for implementation purposes. Taking the largest media unit of a stream as an estimate wastes a lot of memory, especially when using MPEG compression. Sophisticated solutions of mapping are subject of future work. In the following, we will use the term buffer slot to denote the *buffer space* for one media unit.

Since processing time, e.g., for protocol actions does not concern the actual synchronization problem, we will neglect it, whereas an implementation must take it into account. Finally, we assume that control messages are reliably transferred.

Model parameters

n	number of server nodes in the server array	
N	number of media units of a stream	
i, j, υ	media unit index $i, j, \upsilon = 0, ..., N-1$	
k	server index $k = 0, ..., n-1$	
I_j	index set of n subsequent media units starting with media unit j	
S_k	denotes server node k providing substream k	
D	denotes the destination or client node	
s_i	initial sending time of media unit i in server time	[s]
s_i^c	synchronized sending time of media unit i in server time	[s]
a_i	arrival time of media unit i in client time	[s]
d_i	roundtrip delay[10] for media unit i measured at the client	[s]
$d^{\max}$	maximum roundtrip delay	[s]
t_{start}	starting time of the synchronization protocol	[s]
t_{ref}	reference time for the start-up calculation	[s]
t_0	earliest possible playout time of the first media unit	[s]
t_i	expected arrival of the media unit i at the client	[s]
δ_{ij}	arrival time difference between media unit i and j	[s]

A set of media units that needs to be played out at the same time is referred to as *synchronization group*.

[9] We will also use the abbreviation *mu* for media unit.

[10] The roundtrip delay comprises the delay for a control message that requests a media unit and the delay for delivering the media unit.

We assume that media units are distributed in a *round-robin fashion* across the involved server nodes. Hence, we can identify the storage location of a media unit by its media unit number[11], i.e.,

Server $S_{i \bmod n}$ stores the media unit i. (1)

The leads to the following formulation of the *synchronization problem*.

The client must playout the media units of all subsets I_j, with $j \bmod n = 0$, at the *same time*.

2.4 Model 1: start-up synchronization

2.4.1 Introduction

Under the assumption of constant delay and zero jitter, we solve in model 1 the synchronization problem by assuring that the first n media units, which form a synchronization group, arrive at the *same time* at the client. We therefore *need*

$$t_i = t_0, \quad \forall i \in I_0 . \tag{2}$$

The major problem addressed by model 1 is the compensation for different delays due to the independence of the different substreams. For instance, the geographical distance from server to client may be different for each server. Thus, starting transmission of media units in a synchronized order would lead to different arrival times at the client, with the result of asynchrony. Usually, this is compensated by delaying media units at the client [Esc94]. Depending on the location of the sources, large buffers may be required.

In order to avoid buffering to achieve the equalization of different delays, we take advantage of the fact that stored media offers more flexibility. The idea is to initiate playout at the servers such that media units arrive at the sink in a synchronous manner. This is performed by shifting the starting times of the servers on the time axis in correlation to the network delay of their connection to the client. The proposed start-up protocol consists of two phases.

- In the first phase, called *evaluation phase*, roundtrip delays for each substream are calculated, while
- in the second phase, called *synchronization phase*, the starting time for each server is calculated and transmitted back to the servers.

The model is based on the assumption of a *constant end-to-end delay* without any jitter. We further exclude, for the moment, changing network conditions, server drop-outs, and *clock drift*. In such a scenario, synchronization needs to be done once at the beginning and is maintained automatically afterwards.

We need to introduce some more notation to express inter-dependencies between the parameters of the model. We then give a description of the start-up protocol flow and prove its correctness. We close the section with an example of the protocol.

[11] This implies that each substream will send media units at the *same rate*. An extension of the scheme to different media unit rate, each one being the integer multiple of a base rate, is straightforward.

The starting time t_{start} of the protocol equals the beginning of the first phase. Without loss of generality, let

$$t_{\text{start}} = 0 . \tag{3}$$

To begin with, we regard the first n media units of a stream given by I_0 that are distributed across the n servers. The roundtrip delay d_i for the media unit i is given by the difference between its arrival time a_i and the starting time of the synchronization protocol:

$$d_i = a_i - t_{\text{start}}, \quad \forall i \in I_0 . \tag{4}$$

The second phase of the protocol begins at time t_{ref}, which is determined by the last of the first n media units that arrives.

$$t_{\text{ref}} = \max\{a_i | i \in I_0\} \tag{5}$$

The difference between the arrival times of arbitrary media units i and j is needed to calculate the starting times of the servers. We define the difference as follows:

$$\delta_{ij} = a_i - a_j \quad \forall i, j \tag{6}$$

2.4.2 Start-up protocol

The synchronization protocol for starting playback at the server is launched after all involved parties are ready for playback. It can be divided into two phases: *evaluation phase* and *synchronization phase*. The goal of the first phase is to compute the roundtrip delays d_i, $\forall i \in I_0$, while the second phase calculates the starting times and propagates them back to the servers. During start-up, the client sends two different kinds of control messages to the servers:

- *Eval_Request*(i): client D requests media unit i from server S_i, $\forall i \in I_0$.
- *Sync_Request*(i, s_i^c): client D transmits the starting time s_i^c to server S_i.

(a) Evaluation phase

- At local time t_{start}, client D sends an *Eval_Request*(i) to servers S_i, $\forall i \in I_0$.
- Server S_i receives the *Eval_Request*(i) at local time s_i, $\forall i \in I_0$.
- Server S_i immediately sends media unit i time-stamped with s_i back to client D, $\forall i \in I_0$.
- At local time a_i, client D receives media unit i from server S_i, $\forall i \in I_0$.
- At local time t_{ref}, client D has received the last media unit.
  The roundtrip delays $d_i = a_i - t_{\text{start}}$, $\forall i \in I_0$, and the maximum roundtrip time as $d^{\max} = \max\{d_i | i \in I_0\}$ are computed.

(b) Synchronization phase

- At local time t_{ref}, client D computes t_0 as $t_0 = \max\{t_{\text{ref}} + d_i | i \in I_0\}$, the index v that determines t_0 as $v \in I_0$, with $t_{\text{ref}} + d_v = t_0$, and the delay differences as $\delta_{vi} = a_v - a_i$, $\forall i \in I_0$[12]

[12] In case of subframe striping, δ_{vi} may be reduced to $\delta_{vi} = d^{\max} - d_i$, $\forall i \in I_0$. We need a double index for δ_{vi} to cover also inter-frame striping, where the index of $d^{\max}$ must not necessarily equal v.

- With these results, the starting time of server S_i is calculated in server time $s_i^c = s_i + d^{\max} + \delta_{vi}, \forall i \in I_0$.
- Client D sends a $Sync\_Request(i, s_i^c)$ to server S_i, $\forall i \in I_0$.
- At local time $s_i + d_i(t_{\text{ref}} - a_i)$, server S_i receives the $Sync\_Request(i, s_i^c)$, $\forall i \in I_0$.
- At local time s_i^c, server S_i starts scheduling of the substream by sending media unit i, $\forall i \in I_0$.[13]
- At local time t_i, client D receives media unit i, $\forall i \in I_0$.

At any time, only one synchronization group of n media units must be buffered at the client; after the complete reception, the media units are played out immediately. To show the correctness of our mechanism, we first discuss the calculation of the earliest possible playout time t_0 for the first media unit as stated in Theorem 1. The future starting times s_i^c are given according to Theorem 2.

Calculation of the earliest possible playout time t_0 for the first media unit

We need to choose t_0 such that all media units $i \in I_0$ can be delivered and played out in time, i.e., they will arrive at their deadline t_i, which is given by (2) as $t_i = t_0$. Obviously, media unit $i \in I_0$ delivered by server S_i cannot be expected earlier than $t_{\text{ref}} + d_i$. Hence, the substream with the largest delay determines t_0.

Theorem 1. *Let $t_0 = \max\{t_{\text{ref}} + d_i \mid i \in I_0\}$. Then all media units can be delivered and played out in time.*

Proof. Since the earliest possible arrival time for media unit $i \in I_0$ is $t_{\text{ref}} + d_i$, we need to show that $t_i \geq t_{\text{ref}} + d_i, \forall i \in I_0$.
Let $t_0 = \max\{t_{\text{ref}} + d_i | i \in I_0\}$
(2) $\Rightarrow t_i = t_0 \geq t_{\text{ref}} + d_i, \forall i \in I_0$,
$\Rightarrow t_0$ does not violate the arrival times of other substreams.
To show that t_0 is minimal, we assume that $\exists\, \tilde{t}_0 < t_0$
$\Rightarrow \exists\, i_0$, with $\tilde{t}_0 < t_{\text{ref}} + d_{i_0}$
(2) $\Rightarrow \tilde{t}_0 = t_{i_0} < t_{\text{ref}} + d_{i_0}$
$\Rightarrow$ contradiction to the earliest possible arrival time.

For the calculation of the future starting times s_i^c, $\forall i \in I_0$, we define the substream v determining t_0 as follows:

$$v \in I_0 \text{ with } t_{\text{ref}} + d_v = t_0. \tag{7}$$

Calculation of the synchronized sending time s_i^c of media-unit i for server S_i

Substream v is considered *critical* since it determines the starting times of all other initial substreams, and is therefore considered as a reference point to which all other substreams are adjusted. Clearly, the future starting time s_i^c of substream i is composed of the initial starting time s_i plus the maximum roundtrip delay $d^{\max}$. This sum is corrected by the relative arrival time distance δ_{vi} between media unit i and media unit v. This gives the starting time that provides a simultaneous arrival of the media units of any substream i and the media units of substream v. The calculation based on (8), (7), (4) is stated in the following theorem[14].

[13] We are retransmitting the initial media units in the synchronization phase in order to have exactly the same conditions as in the evaluation phase. This guarantees correctness of the scheme also in the case of interframe striping, where media units may have different size.

[14] One can easily imagine situations where synchronization is needed not only at the beginning of a stream. A typical example is the VCR function "pause". After having paused, it becomes necessary to resynchronize again, starting with the media unit subsequent to the last one displayed. The described scheme can be generalized to any series of subsequent media units requested by the client [Gey95].

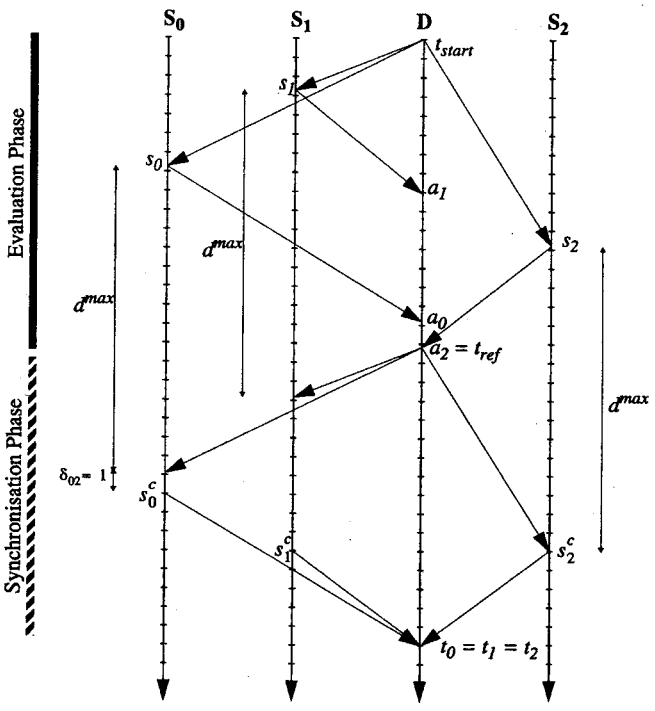

Fig. 6. Example of the start-up synchronization protocol flow

Table 1. Example for the start-up calculations

Server	a_i	d_i	t_{ref}	δ_{2i}	s_i^c
S_0	11	11	12	1	$s_0 + 13$
S_1	6	6	12	6	$s_1 + 18$
S_2	12	12	12	0	$s_2 + 12$

Theorem 2. *Let $s_i^c = s_i + d^{\max} + \delta_{vi}, \forall i \in I_0$. Then media unit $i \in I_0$ will arrive at client time t_i.*

Proof. For each $i \in I_0$: at client time t_{ref}, s_i^c is sent back to server S_i, which receives it at server time $s_i + d^{\max}$ (see Fig. 6). If S_i sent media unit i immediately back to D, it would arrive at client time $t_{\text{ref}} + d_i$. The term $s_i + d^{\max}$ is corrected by δ_{vi}.
Media-unit i will arrive at D at client time
$t_{\text{ref}} + d_i + \delta_{vi} = t_{\text{ref}} + (a_i - t_{\text{start}}) + a_v - a_i)$
(3) $= t_{\text{ref}} + a_v = t_{\text{ref}} + d_v$
(Theorem 1) $= t_0$
(2) $= t_i$

2.4.3 Example of the start-up protocol

The following example in Fig. 6 illustrates model 1. We assume $n = 3$, i.e., three servers, with one substream each. The calculated starting values are shown in Table 1.

For each server and for the client D, a time axis is provided. Arrows indicate control messages or media units, respectively, that are transferred between client and servers. With $t_0 = \max\{t_{\text{ref}} + d_i | i \in I_0\} = \max\{23, 18, 24\} = 24$, we get $v = 2$.

Substream 2 experiences the largest roundtrip delay d^{max}, and therefore determines t_{ref}. Substream 2 is critical because it cannot be started earlier than $s_2 + 12$. As indicated on the time axis for server 0, substream 0 could be started earlier, but is delayed to arrive at the same time as substream 2.

2.5 Model 2: intra- and inter-stream synchronization

2.5.1 Introduction

Model 1 shows how to cope with different but constant delays for each substream originating from a single server node. However, synchronization is performed under the assumption that jitter does not exist. Model 2 loosens this assumption and takes into account *end-system jitter* and *network jitter*. We consider the cumulative jitter to which the factors described in Sect. 2.2 contribute, and we assume that the jitter is bounded.

When subject to jitter, media units will not arrive in a synchronized manner, although they have been sent in a timely manner. The temporal relationship within one substream is destroyed and time gaps between arriving media units vary according to the occurred jitter. Thus, an isochronous playback cannot be achieved if arriving media units of a substream would be played out immediately. Furthermore, jitter may distort the relationship between media units of a synchronization group. Hence, *intra-stream synchronization* as well as *inter-stream synchronization* is disturbed. To smooth out the effects of jitter, media units must be delayed at the sink such that continuous playback can be guaranteed. Consequently, *playout buffers* corresponding to the amount of jitter are required.

The main point addressed by model 2 is the intra- and inter-stream synchronization and the calculation of the required buffer space. First, we regard the synchronization of a single substream. Based on a rule of Santoso et al. [San93], we formulate a theorem that states a well-defined playout time (the playout time or playout deadline is defined as the time elapsed at the client between arrival and playout of the first media unit of a substream) for a substream such that intra-stream synchronization can be guaranteed. Using this so-called *playout deadline*, we derive the required buffer space. Smooth playout cannot be guaranteed if starting before the playout deadline. Starting at a later time would require more buffer space. Afterwards, we will extend our considerations to the synchronization of multiple substreams. The main idea in order to achieve inter-stream synchronization is to maintain intra-stream synchronization for each substream [Ish95]. Each one of the substreams is assumed to have a different jitter bound. In this case, buffer reservation according to a single substream is not sufficient anymore, as inter-stream synchronization will be disturbed for the reason of differences in the jitter bounds. Additional buffering is required to compensate for this. Finally, we examine the effects of the start-up protocol (model 1) on buffer requirements in the case of jitter. The application of model 1 to initiate playback of the servers in a synchronized manner can introduce an error due to jitter. We give a worst case estimate for the error and additional buffer requirements are computed accordingly.

We begin with an extension of the model parameters used so far.

Model parameters

m	substream or server index, $m = 0, \ldots, n-1$	
r	requested display rate of each substream at the client	[mu/s]
d_k^{max}	maximum delay for substream k	[s]
d_k^{min}	minimum delay for substream k	[s]
$\bar{d}_k$	average delay for substream k	[s]
Δ_k	jitter for substream k	[s]
Δ^{max}	maximum jitter of all substreams	[s]
Δ_k^+	maximum upper deviation from $\bar{d}_k$ due to jitter for substream k	[s]
Δ_k^-	maximum lower deviation from $\bar{d}_k$ due to jitter for substream k	[s]
Δ^{max+}	maximum upper deviation of all substreams	[s]
b_k	buffer requirement for substream k at the client	[mu]
b_k^S	buffer requirement for substream k at the client with shifting	[mu]
b_k^M	buffer requirement for substream k at the client with max. jitter	[mu]
B	total buffer requirement for a synchronization group	[mu]
B^S	total buffer requirement for a synchronization group with shifting	[mu]
B^M	total buffer requirement for a synchronization group with max. jitter	[mu]

Throughout this paper, we assume *bounded* jitter and we use the definition of jitter given by Rangan et al. [Ran92], who define jitter as the difference between the maximum delay and the minimum delay.

$$\Delta_k = d_k^{max} - d_k^{min}, \quad \forall k; \tag{8}$$

$$\Delta^{max} = \max\{\Delta_k | k \in \{0 \ldots n-1\}\}. \tag{9}$$

In addition to this, we need jitter bounds that are defined as the deviation from the average delay $\bar{d}_k$. Jitter is in general not distributed symmetrically. Thus, Δ_k^+ and Δ_k^- may not be equal. For further considerations, we assume interdependencies as follows.

$$\Delta_k = \Delta_k^+ + \Delta_k^-, \quad \forall k; \tag{10}$$

$$d_k^{max} = \bar{d}_k + \Delta_k^+, \quad \forall k; \tag{11}$$

$$d_k^{min} = \bar{d}_k - \Delta_k^-, \quad \forall k; \tag{12}$$

$$\Delta^{max+} = \max\{\Delta_k^+ | k \in \{0 \ldots n-1\}\}. \tag{13}$$

2.5.2 Synchronized playout for a single substream

To guarantee the timely presentation of a single stream subject to jitter, it is necessary to buffer arriving media units at the client to compensate the jitter. The buffer is emptied at a constant rate.

Santoso et al. [San93] have already shown that the temporal relationship within one continuous media stream can be preserved by delaying the output of the first media unit for $d_k^{max} - d_k^{min}$ seconds. Based on this theorem, both the playout deadline and the buffer requirements are derived. The

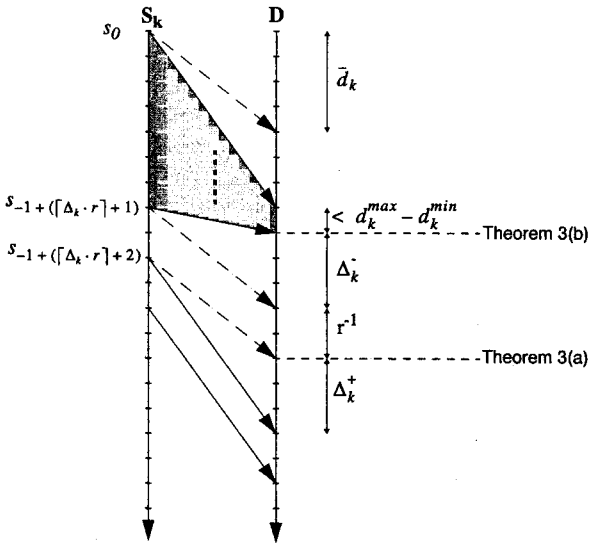

Fig. 7. Worst case scenario for a single substream

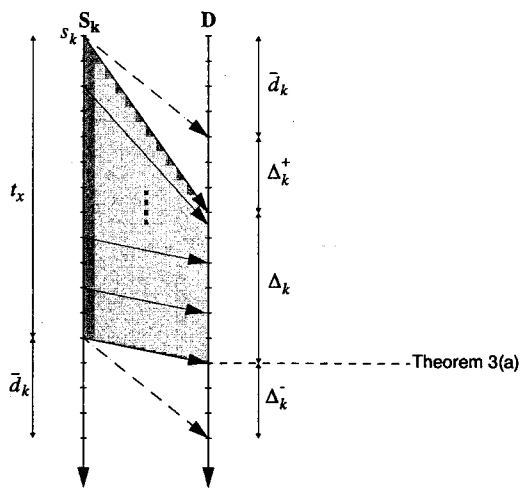

Fig. 8. Worst case scenario for a burst arrival

deadline given by Santoso et al. (case (a)) can be lowered in some situations (case (b)).

Theorem 3. *Consider a single substream k in the case of bounded jitter Δ_k given by (8). Then smooth playout can be guaranteed whenever either one of the following starting conditions holds true.*
(a) $d_k^{\max} - d_k^{\min} = \Delta_k$ seconds elapsed after the arrival of the first media- unit, or
(b) the $(\lceil \Delta_k \cdot r \rceil + 1)$-th media unit has arrived.

Proof. A proof for (a) can be found in [San93]. Condition (b) improves (a) in some cases, i.e., playout can start earlier without violating timeliness, as shown in Fig. 7: The first media unit experiences the maximum delay, subsequent media units arrive in a burst (indicated by the gray area in Fig. 7) such that after the arrival of the $(\lceil \Delta_k \cdot r \rceil + 1)$-th media unit, the elapsed time is less than $d_k^{\max} - d_k^{\min}$. The average delay of media units is denoted by dashed arrows. Assuming that the $(\lceil \Delta_k \cdot r \rceil + 1)$-th media unit has just arrived, we start the playout of the buffered media units immediately. A number of $\lceil \Delta_k \cdot r \rceil + 1$ media units is at least sufficient for a presentation period of $(\lceil \Delta_k \cdot r \rceil + 1) \cdot r^{-1} \geq \Delta_k + r^{-1}$ seconds. In the worst case, the $(\lceil \Delta_k \cdot r \rceil + 1)$-th media unit experiences its minimum delay and the subsequent media unit its maximum delay. Then the maximum period without any arrival is given by $\Delta_k^- + r^{-1} + \Delta_k^+ = \Delta_k + r^{-1}$ seconds. $(\lceil \Delta_k \cdot r \rceil + 1) \cdot r^{-1}$ gives an upper bound to $\Delta_k + r^{-1}$. Consequently, the next media unit arrives just in time. Following media units will not arrive later because the last one has already experienced the largest delay.

Theorem 4 enables us to calculate the maximum required buffer space for the synchronization of a single substream.

Theorem 4. *Consider a single substream k in the case of bounded jitter Δ_k. Let Theorem 3 be applied as a static condition. Then a buffer space of at most $\lceil 2\Delta_k \cdot r \rceil$ media units is required to guarantee intra-stream synchronization.*

Proof. To begin with, we regard the rule of Santoso et al. stated in Theorem 3 (a). We derive the required buffer space based on a worst case scenario outlined in Fig. 8.

Let the first media unit of substream k arrive with his maximum delay, while all subsequent media units experience their minimum delay. Assuming that no media unit can overtake another, in this worst case, a burst of media units occurs before the deadline of Δ_k given by Theorem 3 (a) is reached. The situation is indicated by the gray area in Fig. 8. The time t_x elapsed between the first media unit of the burst and the last one that arrives just at the beginning of the playout computes as follows:

$$t_x = \bar{d}_k + \Delta_k^+ + \Delta_k + \Delta_k^- - \bar{d}_k = 2\Delta_k.$$

Thus, in the worst case, the client must buffer $2\Delta_k$ seconds of playback time, corresponding to $\lceil 2\Delta_k \cdot r \rceil$ media units. Further buffering is not needed because subsequent media units cannot arrive earlier.

Theorem 3 (b) improves the rule of Santoso et al., as indicated in the proof for Theorem 3. Clearly, playout can start before Δ_k seconds have elapsed as shown in Fig. 7. But in such a situation, the rule of Santoso et al. is equally applicable, resulting in a later beginning of the playout, i.e., the consumption from the buffer would start later. We have already covered the worst case for Theorem 3 (a), and therefore conclude that, by employing Theorem 3 (b), no further buffering beyond $2\Delta_k$ seconds of playback time is required.

2.5.3 Synchronized playout for multiple substreams

The basic idea of the synchronization scheme in model 2 is to achieve inter-stream synchronization between multiple substreams by intra-stream synchronization. Once intra-stream synchronization has been established by satisfying Theorem 3 and 4 for each substream, inter-stream synchronization is attained [Ish95; San93]. This holds true if each substream would experience the *same* jitter.

In the following, we consider different jitter bounds for each substream and examine the impact on buffer requirements. The jitter bounds will differ for the case that the paths from sources to the destination are different. We assume that

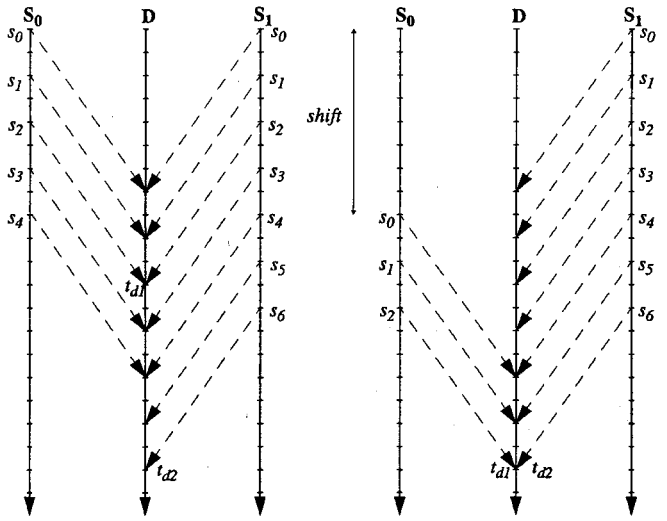

Fig. 9. Multiple substreams without and with shifting

media units experience an average delay $\bar{d}$ on all substream connections. The following proofs can also be carried out with different average delays (see Appendix B for the proof of Theorem 5).
We present two methods to compute the buffer requirements for multiple substreams.

- The first approach estimates the jitter for all substreams with the maximum jitter value.
- The second strategy attempts to refine this coarse-grain estimation by shifting the starting times of each substream in correlation to their jitter values in order to save buffer space.

(a) Maximum jitter strategy

Obviously, playout can only start if Theorem 3 is satisfied for *all* substreams. Thus, the playout deadline for a stream constituted by a synchronization group is defined by the substream that satisfies Theorem 3 last of all. The situation is complicated by different jitter bounds for the corresponding substreams, which lead to different playout deadlines and buffer requirements. We must avoid a situation where substreams with large jitter bounds still wait for their deadlines, while the buffer of other substreams with small jitter bounds already overflows. Ishibashi et al. [Ish95] propose a straightforward solution that allocates a buffer whose size is determined by the substream with the largest jitter bound. Hence, the buffer requirement b_k^M for each substream of the group and B^M for the complete group are given as follows.

$$b_k^M = \lceil 2\Delta^{\max} \cdot r \rceil; \tag{14}$$

$$B^M = \sum_{k=0}^{n-1} b_k^M = n \cdot \lceil 2\Delta^{\max} \cdot r \rceil. \tag{15}$$

(b) Shifting strategy

Depending on the differences in the jitter values of the substreams, the maximum jitter strategy might lead to a buffer waste. A more sophisticated way to handle this problem is to synchronize the different substreams such that they reach their playout deadline *at the same time* on average. This is done by shifting the starting points of all substreams according to the deadline of the substream with the largest jitter bound. Figure 9 depicts such a scenario for two sources, where $\bar{d} = 3.5$, $\Delta_0 = 2$ and $\Delta_1 = 6$.

With Theorem 4, we get buffer requirements of four media units for substream 0 and 12 media units for substream 1. Substream 0 reaches its playout deadline on average at t_{d1} and substream 1 at t_{d2}. Without shifting, a buffer overflow occurs when receiving the 5-th media unit of substream 0, while substream 1 must wait another two time units until playout can start. By shifting, both substreams are due for playout at the same time. The amount of the forward shift can be easily derived from Theorem 3. The k-th substream has to be shifted forward on the time axis with the difference of its jitter to the maximum jitter, i.e., $\Delta^{\max} - \Delta_k$ units of time. Clearly, substream k has to be started $\Delta^{\max} - \Delta_k$ units later than the substream with the largest jitter. When applying that shift, one might conclude that no further buffering is needed, except for the buffer given by Theorem 4. In fact, there is a worst case that requires additional buffer space for each substream. The amount of additional buffering is stated in Theorem 5.

Theorem 5. *Consider a synchronization group consisting of n substreams. With each substream being played out at a rate r. Each substream k has bounded jitter Δ_k and is started $\Delta^{\max} - \Delta_k$ seconds later than the substream with the maximum jitter $\Delta^{\max}$, i.e., a shift of $\Delta^{\max} - \Delta_k$ is applied. Furthermore, substreams are assumed to be synchronized to their average delay prior to the beginning of their playback. Let Theorem 3 be applied. Then a buffer space of $\lceil (\Delta^{\max+} - \Delta_k^+) \cdot r \rceil$ media units for each substream k, in addition to the buffer space required by Theorem 4, is needed to guarantee intra- and inter-stream synchronization for multiple dependent substreams.*

Proof. We assume that all substreams have been shifted according to the substream with the largest jitter. Let m be the substream with the largest jitter $\Delta^{\max}$ and let k be any other substream that has been shifted $\Delta^{\max} - \Delta_k$ seconds. The proof is based on worst case considerations described in Figs. 10 and 11 .

Regard two substreams which reach their playout deadline[15] at the same time on average due to the shifting strategy. The worst case that can happen is that one substream still waits for its playout deadline, while the buffer of the other substream overflows. This becomes true if all media units of one substream experience their minimum delay and all media units of the other substream experience their maximum delay. We aligned the playout deadlines of the substreams on average. So, the latest possible playout deadline is determined by the substream with the largest positive jitter bound Δ^+ if all media units of this substream experience their largest delay. We can therefore distinguish two cases[16]:
1. $\Delta_m^+ \geq \Delta_k^+$ and the first media unit of substream m experiences $d_m^{\max}$, while all media units of substream k arrive with their minimum delay $d_k^{\min}$.

[15] We refer to the playout deadline given in Theorem 3 (a).

[16] Since we assume arbitrary values of Δ_k^+ and Δ_k^-, we need to distinguish two cases in order to show that the additional buffer requirement depends on the substream with the largest upper jitter bound $\Delta^{\max+}$.

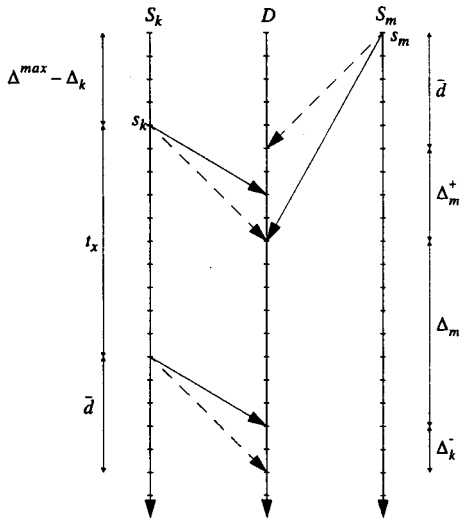

Fig. 10. Worst case scenario 1 for multiple substreams with shifting

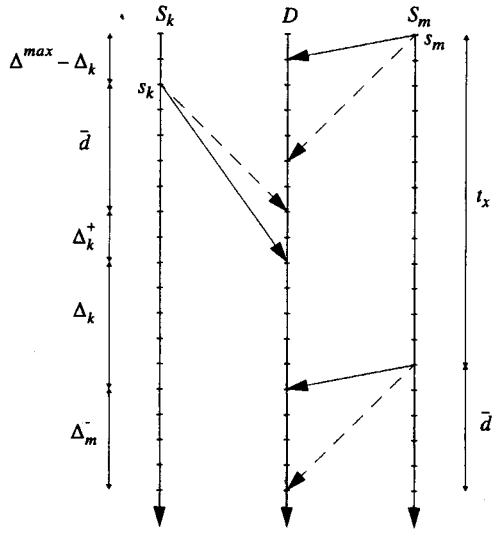

Fig. 11. Worst case scenario 2 for multiple substreams with shifting

2. $\Delta_m^+ < \Delta_k^+$ and the first media unit of substream k experiences $d_k^{\max}$, while all media units of substream m arrive with their minimum delay $d_k^{\min}$.

Case 1.
According to Theorem 3, the playout deadline for substream m is reached at the latest $\bar{d} + \Delta_m^+ + \Delta_m$ seconds after the first media unit of m has been sent. We are interested in the maximum number of media units of the k-th substream that may arrive before the playout deadline is reached. We consider the period t_x between the first media unit of the k-th substream and that media unit of the k-th substream that arrives just at the playout deadline of the m-th substream.

$$\begin{aligned} t_x &= \bar{d} + \Delta_m^+ + \Delta_m + \Delta_k^- - (\bar{d} + \Delta^{\max} - \Delta_k) \\ &= \bar{d} + \Delta_m^+ + \Delta_m^+ + \Delta_m^- + \Delta_k^- \\ &\quad -(\bar{d} + \Delta_m^+ + \Delta_m^- - \Delta_k^+ - \Delta_k^-) \\ &= \Delta_m^+ + 2\Delta_k^- + \Delta_k^+ \end{aligned}$$

Applying Theorem 3, we have already allocated $2\Delta_k$ buffer space in terms of time. From this we conclude an additional buffering of $\lceil(\Delta_m^+ + 2\Delta_k^- + \Delta_k^+ - 2\Delta_k)r\rceil = \lceil(\Delta_m^+ - \Delta_k^+)r\rceil$ media units for the k-th substream.

Case 2.
Case 2 can be shown analogous to case 1 (see Fig. 11). The period t_x is computed as follows:

$$\begin{aligned} t_x &= (\Delta^{\max} - \Delta_k) + \bar{d} + \Delta_k^+ + \Delta_k + \Delta_m^- - \bar{d} \\ &= (\Delta_m^+ + \Delta_m^- + \Delta_k^+ - \Delta_k^-) + \Delta_k^+ + \Delta_k^+ + \Delta_k^- + \Delta_m^-) \\ &= \Delta_k^+ + \Delta_m^+ + 2\Delta_m^- \end{aligned}$$

We have already allocated $2\Delta_m$ buffer space for substream m. Thus, we get additional buffering of $\lceil(\Delta_k^+ + \Delta_m^+ + 2\Delta_m^- - 2\Delta_m)r\rceil = \lceil(\Delta_k^+ - \Delta_m^+)r\rceil$ media units for the m-th substream.

We can conclude that the additional required buffer space does not depend on the substream with the largest jitter, but on that one with the largest upper jitter bound $\Delta^{\max+}$ defined in (13). We can therefore derive an additional buffering of $\lceil(\Delta^{\max+} - \Delta_k^+)\cdot r\rceil$ media units for an arbitrary substream k.

With the above theorems, the total buffer requirements c be computed as follows.

$$b_k^S = \lceil(2\cdot\Delta_k + \Delta^{\max+} - \Delta_k^+)\cdot r\rceil; \quad (1$$

$$B^S = \sum_{k=0}^{n-1} b_k^S = \sum_{k=0}^{n-1} \lceil(2\cdot\Delta_k + \Delta^{\max+} - \Delta_k^+)\cdot r\rceil. \quad (1$$

Theorem 6. *Applying the shifting strategy for the synchi nization of multiple substreams saves buffer space whene* $\Delta_k^- \le \Delta^{\max} - \Delta^{\max+}, \forall k$ *holds true*[17].

Proof. To prove Theorem 6, we must show that $b_k^S \le b$ For the following considerations, we will express the buf requirements in terms of time. Let

$$\hat{b}_k^S = 2\cdot\Delta_k + \Delta^{\max+} - \Delta_k^+, \quad \forall k; \quad \text{and}$$
$$\hat{b}_k^M = 2\cdot\Delta^{\max}, \quad \forall k.$$

Then, $\hat{b}_k^M - \hat{b}_k^S$

$$\begin{aligned} &= 2\cdot\Delta^{\max} - (2\cdot\Delta_k + \Delta^{\max+} - \Delta_k^+) \\ &= 2\cdot\Delta^{\max} - 2(\Delta_k^+ + \Delta_k^-) - \Delta^{\max+} + \Delta_k^+ \\ &= 2\cdot\Delta^{\max} - \cdot\Delta_k^+ - 2\cdot\Delta_k^- - \Delta^{\max+} \\ &= 2\cdot\Delta^{\max} - \Delta_k - \Delta_k^- - \Delta^{\max+} \\ &= 2\cdot\Delta^{\max} - (\Delta_k) - (\Delta_k^- + \Delta^{\max+}) \ge 0, \end{aligned}$$

because $\Delta_k \le \Delta^{\max}$ with (9) and $\Delta_k^- \le \Delta^{\max} - \Delta^{\max+}$.
$\Rightarrow \hat{b}_k^S \le \hat{b}_k^M$

To demonstrate the buffer savings, we computed $B^M - j$ for two substreams. For substream 0, we have chosen a fix jitter value of $\Delta_0 = 40$ ms, while Δ_1 is varied with spect to Δ_0 in steps of 20 ms, taking the values 60, 80, . 200 ms. For each substream, we admitted three values Δ_k^+:

- high: $\Delta_k^+ = 3/4\cdot\Delta_k$,
- medium: $\Delta_k^+ = 1/2\cdot\Delta_k$,

[17] The assumption of $\Delta_k^- \le \Delta^{\max} - \Delta^{\max+}$ holds true for a variet realistic values of Δ_k^- and Δ_k^+, specifically for $\Delta_k^+ = \Delta_k^-$.

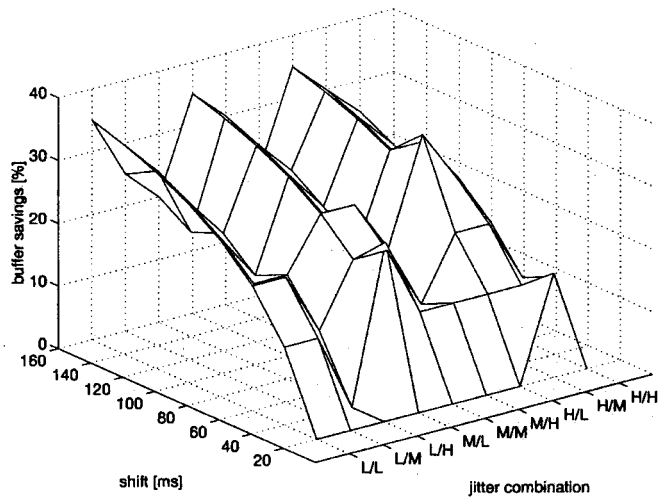

Fig. 12. Buffer saving for different shifts and jitter combinations

– low: $\Delta_k^+ = 1/4 \cdot \Delta_k$.

We calculated the shift of $\Delta_1 - \Delta_0$ according to Theorem 5, and for each of the shift values, we regarded all possible combinations of the Δ_k^+ values between the two substreams.

Combinations are denoted by a pair Δ_0^+/Δ_1^+, where Δ_0^+ and Δ_1^+ can take the values H, M, and L for high, medium, and low, e.g., (H/M). Figure 12 depicts the buffer savings for the different values and jitter combinations. The numerical values for the buffer savings are provided in the appendix in Table 4.

For an arbitrary combination of jitter values, buffer savings increase when the shift between the two substreams becomes large, that is, the larger the difference in jitter values between substreams, the more buffer is saved due to Theorem 5. For a shift of 160 ms for instance, buffer savings up to 35% compared to the maximum-jitter strategy are possible. If the difference in jitter between two substreams equals zero, both the maximum jitter strategy and the shifting strategy require equal buffering.

For the combinations of jitter values, a wave form can be observed in Fig. 12. The lower the maximum value Δ^{max+}, the more buffers are saved (compare proof for Theorem 6). Since Δ_0 is always smaller than Δ_1, Δ^{max+} gets minimal for low Δ_1^+ values. Thus, we obtain the largest buffer savings with the combinations L/L, M/L, and H/L.

2.6 Start-up protocol influence

Until now, we have assumed that substreams are synchronized with respect to their average delay. Model 1 is based on the assumption of zero jitter. When we use the scheme in the case of bounded jitter, we cannot guarantee the synchronization of the substreams with respect to their average delay, since it is based on the roundtrip delay values *experienced* by the first n media units. If the experienced delay corresponds to the average delay, then the start-up protocol works correctly. However, the observed delay can be altered due to jitter, hence the calculation introduces an error that must be considered.

The start-up protocol computation is based on a roundtrip delay for a request packet and one or several packets carrying a media unit. However, the transmission of the request packet from sink to source and the sending of the media unit back to the client are subject to jitter.

Theorem 7. *Let Theorem 3 and 5 be applied. The start-up protocol of model 1 is used to initiate the playback of substreams. Then intra- and inter-stream synchronization for multiple dependent substreams can be guaranteed if, in addition to Theorem 5, a buffer space of* $\lceil \max\{\Delta_m + \Delta_k^+ - \Delta^{max+} | m \neq k \wedge m = 0 \ldots n-1\} \cdot r \rceil$ *media units for each substream k is allocated.*

Proof. The proof is analogous to the one for Theorem 5, except that an additional shift introduced by the start-up calculation is taken into account (see [Gey95]).

2.7 Exact buffer requirements

Model 2 provides a framework to compute buffer requirements for multiple substreams with different jitter bounds to attain inter-stream synchronization by maintaining intra-stream synchronization. Buffer requirements are given by Theorem 4 and 5. The error introduced by the start-up protocol is corrected by Theorem 7. Throughout all theorems, we expressed the time we need to buffer in terms of media units. The required buffer space can be optimized by adding up the time to buffer given by Theorems 4, 5 and 7 and by transforming the resulting sum into buffer slots. Hence, we can sum up the overall buffer requirements b_k for a substream k, and B for a synchronization group consisting of n substreams:

$$b_k = \lceil (2\Delta_k + \Delta^{max+} - \Delta_k^+ + \max\{\Delta_m + \Delta_k^+ - \Delta^{max+} | \; m \neq k \wedge m = 0 \ldots n-1\} \cdot r \rceil; \quad (18)$$

$$B = \sum_{k=1}^{n} b_k \quad (19)$$

$$= \sum_{k=1}^{n} \lceil (2\Delta_k + \Delta^{max+} - \Delta_k^+ + \max\{\Delta_m + \Delta_k^+ - \Delta^{max+} | \; m \neq k \wedge m = 0 \ldots n-1\} \cdot r \rceil.$$

3 Model 3: resynchronization

3.1 Introduction

Models 1 and 2 assured both intra-stream synchronization and inter-stream synchronization under the assumption that jitter is bounded. In ATM-based networks, this assumptions typically holds true at least for the network, because we can express the acceptable QoS in parameters like throughput, delay, jitter or cell losses [Scr92]. If the endsystem is not using a real-time operating system, bounded jitter cannot be guaranteed.

However, when jitter is unbounded, an application needs to make certain assumptions on the amount of jitter, since buffer space may be limited or the increase in end-to-end delay by buffering is unacceptable [Scr92].

Model 3 can be characterized as a scheme for *resynchronization*. We apply the concept of a *buffer-level control*

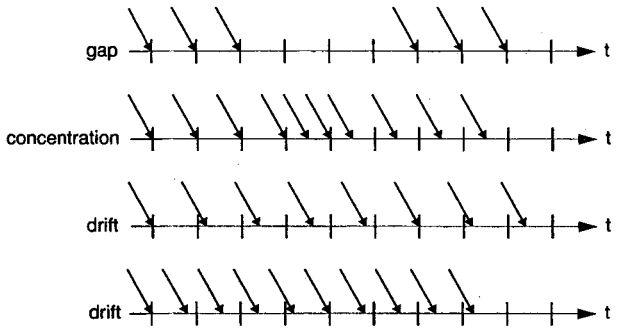

Fig. 13. Different types of disturbances

to detect asynchrony. To recover from asynchrony, we use feedback messages to the servers. Model 3 copes with asynchronies introduced by

- alteration of the average delay,
- clock drift, and
- server drop-outs.

An alteration of the average delay leads to a *gap*[18] or a *concentration* in the continuous media stream. A gap occurs when the average delay becomes larger, a concentration can be observed when the average delay becomes smaller. The situation is illustrated in Fig. 13. The playout intervals at the client are marked on the time axis. Arriving media units are represented by arrows. Notice that an alteration of the average delay is assumed to be of long-term effect, otherwise a disturbance is already covered by the normal buffering due to jitter. A gap or a concentration leads to a shifting of the average buffer level obtained after the substream has been started by employing model 1. Hence, intra-stream synchronization and consequently inter-stream synchronization is disturbed if we assume the jitter bounds to remain constant. Depending on the extent of the shifting, a rising number of lost media units up to total buffer starvation/overflow may be observed.

The result of clock drift is very similar to the result of a change in delay, but arises much more slowly. Clock drift introduces a skew as defined in Sect. 1.3. If a server clock is faster than the client clock (determining the consumption rate), the scheduling frequency will be higher at the server than at the client. Thus, regarding an arbitrary time interval, the arrival rate is higher than the consumption rate. This process accumulates and leads to a buffer overflow. In contrast, if the server clock is slower than the clock at the client, the scheduling frequency will be lower at the server than at the client, resulting in buffer starvation.

A mechanism is needed to adapt to changing conditions, in order to preserve synchronization without allocating additional buffer space. Solving the problem by additional buffering based on worst case estimates might turn out to be a difficult task because changing conditions are unpredictable. Even if we succeed in obtaining worst case estimates, we must be aware that, first, resources are limited and that, second, large playout buffers increase the overall end-to-end delay, which is not desired. Furthermore, uncontrolled buffering compensates the problems to a certain amount, but will not resolve them over a long period of time.

[18] The effect of a server drop-out is also a gap in the media stream.

We have shown that all the described disturbing factors affect the buffer level. Thus, the buffer level can be regarded as an indicator for upcoming synchronization problems. Once a sink has discovered an asynchrony, it must take measures to restore synchronization. As asynchrony is basically a shifting in the media stream, we only need to correct this shifting. Corrective actions must be fed back either to the source or to the sink in order to restore synchrony. The idea of taking the buffer level as an indicator is often referred to as *buffer level control*. Basic work in this area can be found in [Rot95], [Koe94] and [Lit92]. Our model will pick up some of their ideas and extend them to an applicable solution for the synchronization problem. In contrast to their work, we take model 1 and 2 as a basis for synchronization and extend them with a buffer level control. We focus mainly on buffer requirements and parameter tuning.

The next section gives an overview of the used parameters, afterwards models 1 and 2 are examined with respect to a buffer level control, that is, we present a buffer model suitable to realize a buffer level control. Finally, we discuss the degree and the duration of resynchronization actions.

Model parameters

UW_k	upper buffer watermark for substream k	[mu]
LW_k	lower buffer watermark for substream k	[mu]
b_k^A	additional buffer slots for substream k	[mu]
B_k	total buffer size of substream k	[mu]
q_{tk}	queue size of substream k at time t	[mu]
$\bar{b}_{tk}$	smoothed buffer level of substream k at time t	[mu]
o_{tk}	computed resynchronization offset	[mu]
$S(q_{tk})$	smoothing/filtering function	[mu]
$C(\bar{b}_{tk})$	control function	[mu]
$\bar{d}_k^{\text{new}}$	new average roundtrip delay	[s]
R	length of the resynchronization phase	[s]
α	smoothing factor	

3.2 Buffer level control

3.2.1 System model

The concept of buffer level control is often referred to as a *control loop* [Koe94]. Sources transfer media units over the network, the media units arrive at the sink and are buffered before playout. The current buffer level is periodically measured, and if an ill buffer level is found, the appropriate steps are taken. Actions may affect either the buffer itself or the server. In the former case, the loop is placed in the client, in the latter case, it includes the client, the server and the network. Koehler et. al and Rothermel et al. [Koe94; Rot95] propose a synchronization scheme that does not adapt the playout behavior of the server. Actions are taken exclusively at the sink, by changing the consumption rate or by skipping/pausing. This kind of control loop compensates for disturbances to a certain extent, depending on the allocated, available buffer space, but sacrifices the real-time stream continuity.

We adopt a concept where all components of the video server architecture are included in the control loop, similar to the approach of Cen et al. [Cen95]. As shown in Fig. 14,

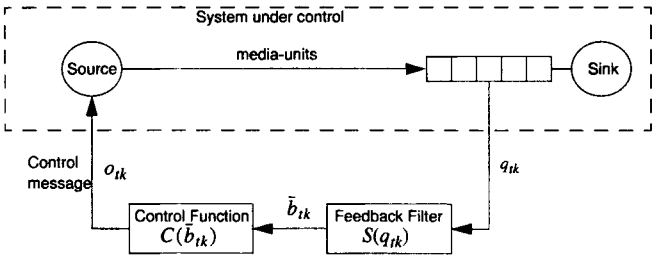

Fig. 14. System model for the buffer level control

the architecture applies feedback actions to the sources via control messages in order to maintain synchronization at the sink.

(a) Feedback filter

The buffer level for substream k at time t is denoted by q_{tk}. This value is periodically passed to a *filtering function* $S(q_{tk})$ to filter short-term fluctuations caused by jitter and to compute the *smoothed buffer level* $\bar{b}_{tk}$. Examples for filtering functions are the geometric weighting smoothing function (with α as smoothing factor) [Rot95; Cen95; Mas90]:

$$\bar{b}_{tk} = S(q_{tk}) = \alpha \cdot \bar{b}_{t-1k} + (1 - \alpha) \cdot q_{tk} (\text{with } \alpha \in [0, 1]). \quad (20)$$

The main goal of filtering is to distinguish between buffer level changes caused by jitter and long-term disturbances. If the filter is too sensitive, or no filter is used at all, jitter causes actions for resynchronization, although no exceptional situation has occurred. On the other hand, a filter that reacts to slowly to changing conditions takes actions too late, with the result of a longer period of buffer starvation or overflow. Thus, presentation quality suffers.

(b) Control function

The smoothed buffer level $\bar{b}_{tk}$ is passed to a *control function* $C(\bar{b}_{tk})$ that takes appropriate actions. For each substream buffer, a *lower watermark* LW_k and an *upper watermark* UW_k are defined. When $\bar{b}_{tk}$ falls below LW_k or exceeds UW_k, there arises the risk of starvation or overflow, respectively, producing an asynchrony. If this happens, a *resynchronization* or *adaptation phase* is entered, whose purpose is to move $\bar{b}_{tk}$ back into between LW_k and UW_k. Depending on the extent of asynchrony, the control function sends an *offset* o_{tk} to the source. The source either skips the number of media units specified in the offset or pauses for a duration of o_{tk} media units. We prefer this technique to an alteration of scheduling speed, respectively production rate, at the source because we think the latter is too resource-demanding. The QoS of other clients serviced by the video server might suffer.

The sink stays in its resynchronization phase for a time R in order to let the smoothed buffer level react on the taken measures. At the end of the resynchronization phase, $C(\bar{b}_{tk})$ controls again whether or not the buffer level has moved back into the normal area between LW_k and UW_k. If not, a new resynchronization phase is started [Rot95].

3.2.2 Buffer requirements and filter tuning

Models 1 and 2 provide the buffer space b_k needed to compensate jitter. In the following consideration, we will denote

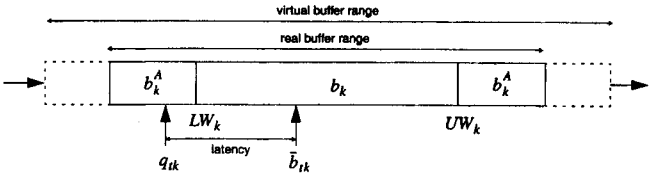

Fig. 15. Buffer model with virtual and real buffer

b_k as *kernel buffer*. Applying a buffer level control only to this buffer is not sufficient, since each buffer level within the range of b_k must be regarded as normal due to jitter effects. We fix LW_k and UW_k to 1 and b_k, respectively. To realize a buffer level control, we must admit buffer levels below and above the watermarks. Otherwise, it is impossible to get the smoothed buffer level below or above the watermarks.

We suggest the scheme of a so-called *virtual buffer* as indicated in Fig. 15 by the dashed lines. The virtual buffer includes at least the real buffer comprising the kernel buffer b_k and an additional buffer b_k^A. The virtual buffer is exclusively used for the calculation of buffer levels below and above the real buffer. This allows for a faster reaction of the smoothing function $S(q_{tk})$. The mapping between the real buffer level and the virtual buffer level q_{tk} is performed as follows.

- If neither buffer starvation nor buffer overflow occurs, the real buffer level equals the virtual buffer level.
- If a buffer overflow occurs, then the virtual buffer is increased for each discarded media unit, while the real buffer levels remains unchanged.
- If a buffer starvation occurs, then the virtual buffer is decreased each time when the client scheduling finds an empty buffer, while the real buffer level remains unchanged.
- If the normal state of the real buffer is restored by resynchronization measures, the virtual buffer level is reset to the real buffer level.

The size of b_k^A strongly influences the gracefulness of the resynchronization[19]. The smoothed buffer level $\bar{b}_{tk}$ always has a latency (see Fig. 15) compared with the virtual buffer level q_{tk}, i.e., q_{tk} might be below LW_k, while $\bar{b}_{tk}$ still needs some time to fall below. Let $b_k^A = 0$, for instance. Then buffer starvation occurs before it is recognized by the control function. Hence, presentation quality suffers depending on the value of b_k^A. We consider the following three cases for the size of b_k^A.

1. Selecting $b_k^A = 0$ yields no gracefulness at all. Asynchrony immediately affects presentation quality and is soon discovered by a viewer.
2. b_k^A can be dimensioned such that a least the period between the rise of asynchrony and the discovery by the control function is covered.
3. For full gracefulness, b_k^A has to be chosen such that asynchrony does not affect presentation at all. The buffer

[19] Notice that the start-up latency is also influenced by the size of b_k^A. The larger b_k^A is, the longer it takes until the first media unit of a substream is played out because the buffer level must exceed LW_k before the playout deadline given by model 2 can be applied.

space has to cover the period between rise, discovery and removal of asynchrony.

3.2.3 Parameter tuning

In our model, we have several parameters that must be chosen appropriately in order to trade off reactiveness and overhead.

(a) Smoothing parameter α
Obviously, the latency of reaction to an asynchrony problem depends strongly on the behavior of $S(q_{tk})$. The more indolently $S(q_{tk})$ reacts, the later a resynchronization phase is entered, the more buffer space b_k^A may be desired to compensate for asynchrony as much as possible. On the other hand, the more sensitively $S(q_{tk})$ reacts, the more often resynchronization is done unnecessarily (due to the effect of jitter), the less buffer space b_k^A is needed to provide sufficient gracefulness. Hence, the tuning of $S(q_{tk})$ is a trade-off between stability and reactivity. The choice or the tuning of $S(q_{tk})$, respectively, helps to determine the additional buffer space b_k^A.

For further consideration, we examine the filtering function given by (20) with respect to the second case described above, i.e., the size of b_k^A must cover the period between rise and discovery of an asynchrony. This case is most interesting, because it is influenced by $S(q_{tk})$. The behavior of the filter is determined by the parameter α.

- A large value of α yields strong smoothing, a stronger consideration of the past, and a more indolent reaction.
- A small value of α yields weak smoothing, a stronger consideration of the present, and a more sensitive reaction.

An upper bound for the choice of is given by the available memory. A lower bound should be chosen such that starvation/overflow events due to jitter can be distinguished from long-term disturbances. Accordingly, α should be set as high as possible, while considering reasonable buffering. In our experiments (see [Gey95] for details), we found that a value of 0.6 or 0.7 for α is a good compromise with respect to the buffer requirement and the number of necessary resynchronization actions. Values of α between 0.1 and 0.5 lead to a relatively high number of unrequired resynchronization actions. For α values 0.8 and 0.9, there are zero resynchronization actions.

(b) Degree o_{tk} of resynchronization
Resynchronization is performed by sending an offset to the servers. The goal is to move the buffer pointer $\bar{b}_{tk}$ back into the area between UW_k and LW_k such that the situation before the occurrence of the disturbance is restored. The size of the offset o_{tk} can be determined by two different strategies: *fixed offset* or *variable offset*.

Employing the fixed-offset strategy, o_{tk} is set to a constant value. Resynchronization is done slowly in subsequent resynchronization phases until synchronization is restored. The value should not be chosen too high, because resynchronization, e.g., due to clock drift, is in the range of one or several media units. High values could lead to oscillation because the extent of asynchrony is always overestimated.

When applying the variable offset strategy, o_{tk} is varied depending on the extent of the occurred asynchrony. Notice that with the variable-offset strategy several resynchronization phases could also be needed because the time when the offset is calculated (determined by the filtering function) often reflects only a fraction of the total extent of asynchrony. Nonetheless, synchronization is generally restored faster with a variable offset. In Sect. 4.5, we present some experimental results comparing both strategies.

(c) Duration R of resynchronization
The duration of a resynchronization phase is defined by R. After R seconds, the control function once more compares the smoothed buffer level with the watermarks. Again, resynchronization actions may be taken.

R must be chosen sufficiently large, so that the server can perform the resynchronization, that is, the action must already have taken effect on the client. Selecting R too small leads to numerous unnecessary resynchronization phases, where, during each phase, the extent of asynchrony is overestimated. Take, for instance, a buffer overflow. Appropriate resynchronization actions are injected, which can result in buffer starvation due to overestimating the asynchrony. Again, resynchronization is started. Thus, low values of R can result in oscillation. For large values of R, several resynchronization phases are also needed, but the total time of resynchronization becomes unacceptably long. So, in both cases presentation quality might be strongly influenced.

3.3 Experimental results

Based on the prototype implementation of the video server array, we have implemented the proposed synchronization scheme for evaluation purposes. For implementation details, refer to [Ber96] and [Gey95].

The following experiments have been performed on a dedicated SUN Sparc 10 workstation as a client. We used two videos, each one distributed across two servers:

- A "Bitburger" commercial, sampled at a rate of 16 fps (frames/s), with a total length of 462 frames.
- A scene from the production "Seaquest", sampled at a rate of 16 fps, with a total length of 6710 frames.

3.3.1 Intra- and inter-stream synchronization

For the first experiment, we measured the inter-arrival times of frames for the video "Bitburger". The experiment was conducted with two substreams. For substream 0 and substream 1, we measured jitter values of 26 ms and 24 ms, respectively (see Table 2). According to Theorems 4, 5 and 7, two buffer slots are allocated for each substream, including a buffer slot to read a frame from the network. The shifting of 2 ms between the two substreams is negligible. The start-up protocol leads to a start-up latency of 252 ms for the first substream and 314.5 ms for the second substream. These times include an additional, overall charge of 200 ms for processing time. Thus, we can see clearly that the maximum roundtrip delay of 52 ms determines the starting time of the first server. The second server starts exactly

Table 2. Experimental results for intra- and inter-stream synchronization

Number of substream	jitter [ms]	buffer [frames]	start-up latency[a][ms]	roundtrip delay [ms]
0	26	2	252	33
1	24	2	314.5	52

[a] The start-up latency is usually defined as the delay between user interaction and visibile feedback

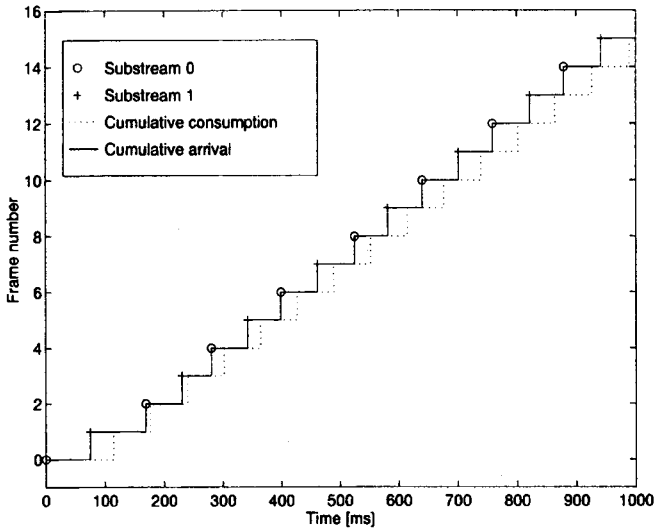

Fig. 16. Cumulative arrival and consumption for two substreams

62.5 ms later, according to the frame rate of 16 fps. These results prove that the start-up protocol is performed with a high accuracy.

Consider now Fig. 16, showing the cumulative arrival times of the first 1000 ms of the "Bitburger" commercial. The x-axis displays the elapsed time while the y-axis shows the frame number. The cumulative arrivals of frames of both substreams never cross the cumulative consumption, i.e., the cumulative arrival always remains above the consumption, indicating that at no time buffer starvation occurred. Thus, the stream is played out smoothly. Further, we can see that at each time only one frame is buffered for each substream at most. This is indicated by the so-called *backlog function*, which states the difference between the cumulative arrival and consumption [Kni95]. In the example, the backlog function takes the value 1 at all times. Consequently, no buffer overflow occurred. The playout deadline given by Theorem 3 is indicated by the beginning of the cumulative consumption. The results show that intra- and inter-stream synchronization is achieved well by employing the synchronization scheme given by models 1 and 2[20].

3.3.2 Buffer-level control

In the second experiment, we evaluated the efficiency of the buffer level control mechanism. Our prototype of the video server array is implemented in an ATM-LAN environment. We thus faced the problem that events like gaps

[20] Notice that the experiment was performed under shifted time access, as described in Sect. 4.1.

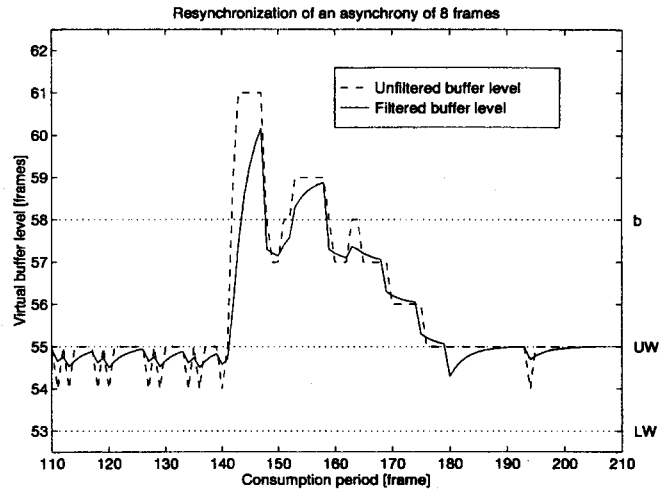

Fig. 17. Resynchronization with fixed offset

or concentrations within a stream are rather unlikely. Therefore, we simulated these events in the servers. The amount of asynchrony can be specified by the user upon starting a server. The server then periodically introduces drop-outs in scheduling or sends several frames at once. The client attempts to resynchronize the server by sending back offsets. We conducted this experiment again with the "Bitburger" commercial striped only onto a single server. The following parameters have been used.

- Smoothing factor for the geometric weighting function: $\alpha = 0.7$.
- Amount of injected asynchrony[21]: -8, -4, $+4$, $+8$
- Resynchronization strategy: *fixed offset* and *variable offset*.

The variable offset was calculated by taking the difference between q_{tk} and the watermarks. The fixed offset was set constantly to 1. In accordance with the previous experiment, we allocated two buffer slots for the substream. This corresponds to the kernel buffer b_k defined in Sect. 3.2. Furthermore, for the additional buffering b_k^A, we were using three buffer slots twice, above and below b_k. Consider now Fig. 17, showing the virtual buffer level and the filtered buffer level over time for the resynchronization of a concentration of eight frames. The y-axis shows the virtual buffer level, while the x-axis denotes the consumption period. The upper bound of the real buffer level is denoted by b, while the lower bound is not shown in the figure. Thus, b_k^A equals $b - UW$ and b_k is given by $UW - LW$. The virtual buffer level ranges from 1 to 108 because we arbitrarily selected a number of 50 frames above and below the real buffer to calculate the virtual buffer. Figure 17 shows the course of resynchronization if the fixed offset strategy is employed.

The first resynchronization phase is entered exactly during consumption period 142, when the filtered buffer level crosses the upper watermark. The virtual buffer level rises up to 61, that is, four frames are discarded. The loss of four frames could be perceived during playback. The client then sends an offset of -1 to the server. After the end of the first

[21] Negative values denote a drop-out, while positive values denote a concentration.

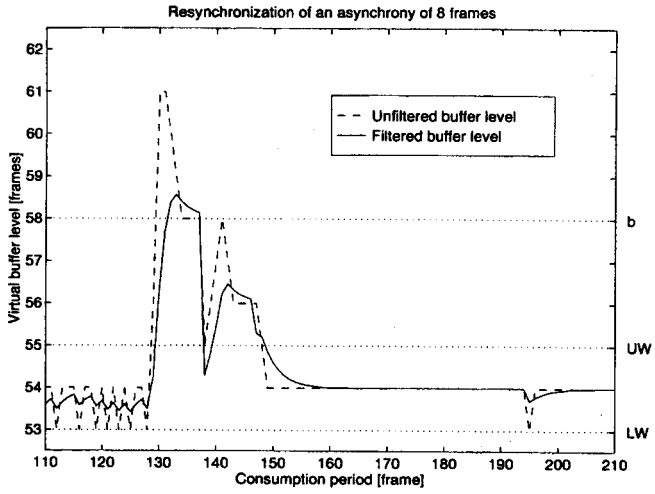

Fig. 18. Resynchronization with variable offset

resynchronization phase, the filtered buffer level is reset to the virtual buffer level, as indicated by the abrupt decrease of the filtered buffer level depicted in Fig. 17. The client undergoes seven subsequent resynchronization phases in all. These phases are indicated by the peaks. Synchronization is restored exactly during consumption period 180, when the filtered buffer level falls below UW. We can therefore conclude a total duration of 38 consumption periods for the resynchronization of an asynchrony of eight frames with the fixed-offset strategy. This corresponds to 2375 ms.

We now consider the same situation with the variable-offset strategy. The course of the filtered and unfiltered buffer level is depicted in Fig. 18.

Resynchronization starts during consumption period 130. Again, a number of four frames is discarded. The client first sends an offset of -3 frames to the server. Just after this resynchronization action, the buffer level falls below UW for a short period of time. Now, two additional resynchronization phases are undergone until synchrony is restored. In each phase, an offset of -2 is sent to the server. Synchronization is exactly restored during consumption period 149. In contrast to the fixed-offset strategy, only 19 consumption periods are needed to regain synchrony. This corresponds to 1187.5 ms.

To obtain more representative values concerning the total duration of resynchronization in dependency of the applied strategy, we performed a third experiment. This time, we took a video of longer duration. During the playback of "Seaquest", 50 resynchronization are undergone. We estimated the duration by taking the mean value of the duration sum. Table 3 presents the experimental results for different sizes of asynchronies.

When applying the variable-offset strategy, the duration of the synchronization can be reduced by more than half compared to the fixed-offset strategy. For an asynchrony of eight frames, only 43% of the duration of the fixed-offset strategy is needed to regain synchronization. The results also show clearly that resynchronization with a variable offset becomes even more efficient for larger asynchronies, because the adoption is performed faster. For negative asynchronies or gaps, respectively, we can see that the gain with the variable offset is not as high as for positive asynchronies. Th due to the way the asynchrony is produced. While concei tions are introduced by sending a specified amount of da once, gaps are produced by just skipping a number of fra at the server. So, a concentration arises immediately and therefore be removed faster. Gaps arise slowly, because buffer at the client is emptied only during the presenta cycle. Thus, resynchronization for gaps is achieved n slowly when comparing the two strategies.

Table 3. Mean and variance of the resynchronization duration

Asynchrony	Fixed offset		Variable offset		
	Mean [ms]	Variance [ms]	Mean [ms]	Variance [ms]	[%][a]
-4	1,143.80	223.13	773.75	78.95	67.6
4	1,327.50	250.50	707.50	139.72	53.3
-8	1,223.80	310.60	665.00	123.56	54.3
8	2515.00	789.21	1,081.20	202.95	43.0

[a] The percentages compare the two strategies and are calculated with re to the results for the fixed-offset strategy

The conducted experiments prove the effectiveness o buffer level control mechanism. Furthermore, they indi that a variable-offset strategy restores synchrony much f than a fixed-offset strategy.

4 Extension to the synchronization scheme

In our model, we have so far assumed that

- the media units from the different servers, which fo synchronization group, must be played out at the *s* time,
- the substreams from the sever nodes are parts of a *si* stream, e.g., an audio or a video stream.

In the following, we will show how the proposed sync nization protocol can be extended to work even when above assumptions are not met.

4.1 Synchronization under shifted time order

In contrast to the synchronization problem we have so so far, we now assume a *shifted* time order between substreams to be synchronized. We will extend our sch to inter-media unit synchronization where media unit different substreams are played out *subsequently*, and this *modified synchronization requirement*. Hence, we h a temporal relationship like: playout of media unit of stream 0, then playout of media unit of substream 1, playout of media unit of substream 2.

Such a synchronization problem arises, for instance the context of our server array, when entire media units stored on the different server nodes (see Fig. 19).

While we assumed so far a display rate of r of *substream* at the client, we now require a display rat r for the *complete* stream consisting of n substreams. means, that the rate within one substream is defined by while the complete stream at the sink is played out with rate r, i.e., subsequent media unit numbers are expecte arrive with a distance of r^{-1}.

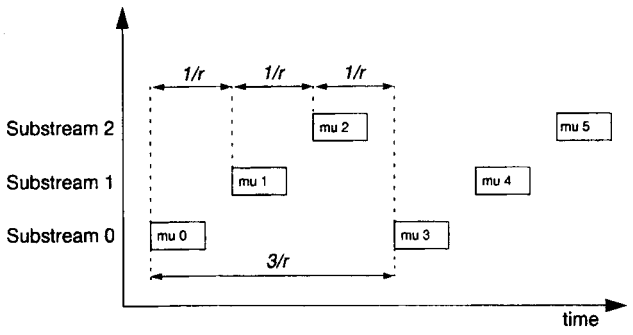

Fig. 19. Temporal relationship for inter-media unit striping

As far as the start-up protocol (model 1) is concerned, we need to make a few adaptations.

The presentation rate r determines the arrival time t_i of media unit i. After the first media unit arrived at t_0, media unit i is expected at time t_i, given by

$$t_i = t_0 + i \cdot r^{-1}, \forall i \; . \tag{21}$$

During the *synchronization phase* of the start-up protocol, the computation of the variables t_0, ν and s_i^c must be modified to take into account that the media units are now expected one after the other. This leads to the following definitions.

- At local time t_{ref}, client D computes $t_0 = \max\{t_{\text{ref}} + d_i - i \cdot r^{-1} | i \in I_0\}$,
- the index ν that determines t_0 as $\upsilon \in I_0$ with $t_{\text{ref}} + d_\upsilon - \upsilon \cdot r^{-1} = t_0$.
- With these results, the starting time of server S_i is calculated in server time $s_i^c = s_i + d^{\max} + \Delta_{\upsilon i} + (i - \nu) \cdot r^{-1}, \forall i \in I_0$.

The proof that the calculation of t_0, s_i^c is correct, which was given in Theorem 1 and 2, can be easily adapted to take into account the modified synchronization requirement.

Theorem 3 defined the conditions for each substream to guarantee a *simultaneous* playout of the media units in a synchronization group. With the modified synchronization requirement, subsequent media units are required to be played out r^{-1} seconds later, relative to their predecessors. To adapt Theorem 3, the idea is simply to *lower* the starting conditions for subsequent substreams according to their distance to substream 0.

Theorem 8. *Consider a synchronization group consisting of n substreams, with each substream being played out at a rate r/n. Each substream has a bounded jitter Δ_k, and the concept of a shifted time order is applied. Then, smooth playout of the complete synchronization group can be guaranteed whenever either one of the following starting conditions is satisfied for each substream k:*
(a) $\Delta_k - (k-1)r^{-1}$ seconds elapsed after the arrival of the first media unit of substream k,
(b) the $(\lceil \Delta_k \cdot r/n \rceil)$-th media unit of substream k arrived, and since then $(n - k + 1) \cdot r^{-1}$ seconds have elapsed.

Proof. The first media unit of the first substream is expected to be played out first. Obviously, to guarantee smooth playout, substream 0 has to satisfy Theorem 3. Let k be an arbitrary substream of a synchronization group. The first media unit of k is expected $(k-1)r^{-1}$ seconds after the first media unit of substream 0. We assume that Theorem 3 holds true for the first substream and that at least $\Delta_k - (k-1)r^{-1}$ seconds have elapsed since the first media unit of the substream k has arrived. If the playout of the first media unit is started anew, we will know exactly that the k-th media unit is needed to be played out in $(k-1)r^{-1}$ seconds. The time passed till this moment amounts up to at least $\Delta_k - (k-1)r^{-1} + (k-1)r^{-1} = \Delta_k$ seconds.

With this, we can conclude that Theorem 3 is satisfied for any substream k when applying Theorem 8 (a).

To prove 8 (b), we argue analogously to the proof of Theorem 3. Again, we assume that Theorem 3 is satisfied for the first substream. Moreover, the $(\lceil \Delta_k \cdot r/n \rceil)$-th media unit of substream k has arrived and $(n - k + 1) \cdot r^{-1}$ seconds have elapsed since the arrival. Playout of the first media unit is started instantaneously. An amount of $\lceil \Delta_k \cdot r/n \rceil$ media units is at least sufficient for a presentation of $\lceil \Delta_k \cdot r/n \rceil \cdot (r/n)^{-1} \geq \Delta_k$ seconds. In the worst case, the maximum period between the arrival of the $(\lceil \Delta_k \cdot r/n \rceil)$-th media unit and the $(\lceil \Delta_k \cdot r/n \rceil + 1)$-th media unit equals $\Delta_k + n/r$ seconds. A period of $(n - k + 1) \cdot r^{-1}$ seconds has already elapsed. Furthermore, media units of substream k are expected $(k-1)r^{-1}$ seconds later than the first media unit of substream 0. We can therefore compute the total elapsed time by

$$\begin{aligned} &((n - k + 1) \cdot r^{-1}) + ((k-1)r^{-1}) \\ &= nr^{-1} - kr^{-1} + r^{-1} + kr^{-1} - r^{-1} = n/r \end{aligned}$$

Thus, the worst case is covered and media unit $(\lceil \Delta_k \cdot r/n \rceil + 1)$ will arrive in time. We can conclude that Theorem 3 is satisfied for any substream k when applying Theorem 8(b).

4.2 Synchronization of multiple streams

The problem of synchronizing multiple streams occurs if several related synchronization streams must be synchronized, e.g., an audio stream and a video stream. Each audio or video stream can itself consist of an arbitrary number of substreams. We call each such stream, which consists of multiple substream and must itself be synchronized with other streams, a synchronization group. We assume that the synchronization groups are presented simultaneously, i.e., media unit i of one group at the same time as media unit i of the other groups. To solve this synchronization problem, we suggest to combine all synchronization groups in a *supergroup*.

The calculations of model 2 can be applied to this supergroup without any modifications. The start-up protocol must be modified slightly to take into account that there exists a different starting point t_0 for each synchronization group. Taking the maximum of these t_0 values as the earliest starting time for all groups allows to initiate start-up of the servers in a synchronized fashion.

Synchronizing a supergroup is a bit more complicated if the rates of the synchronization groups are different. In this case, we can get started once in a synchronized manner. However, further synchronizations necessary due to VCR functions cannot be performed with the start-up protocol

Table 4. Computed buffer savings

Δ_0	Δ_1	shift	jitter combination L/L	L/M	L/H	M/L	M/M	M/H	H/L	H/M	H/H	
40	60	20	6	6	6	6	6	6	6	6	6	B^M
			6	6	6	6	6	6	6	5	6	B^S
			0	0	0	0	0	0	0	16.7	0	%
40	80	40	8	8	8	8	8	8	8	8	8	B^M
			7	7	8	6	7	7	7	7	7	B^S
			12.5	12.5	0	25	12.5	12.5	12.5	12.5	12.5	%
40	100	60	10	10	10	10	10	10	10	10	10	B^M
			8	8	9	8	8	9	8	8	9	B^S
			20	20	10	20	20	10	20	20	10	%
40	120	80	12	12	12	12	12	12	12	12	12	B^M
			9	10	10	9	9	10	8	9	10	B^S
			25	16.7	16.7	25	25	16.7	33	25	16.7	%
40	140	100	14	14	14	14	14	14	14	14	14	B^M
			10	11	12	10	11	12	10	11	12	B^S
			28.57	21.43	14.29	28.57	21.43	14.29	28.57	28.57	21.43	%
40	160	120	16	16	16	16	16	16	16	16	16	B^M
			11	12	13	11	12	13	11	12	13	B^S
			31.25	25	18.75	31.25	25	18.75	31.25	25	18.75	%
40	180	140	18	18	18	18	18	18	18	18	18	B^M
			12	13	15	12	13	14	12	13	14	B^S
			33.33	27.78	16.67	33.33	27.78	22.22	33.33	27.78	22.22	%
40	200	160	20	20	20	20	20	20	20	20	20	B^M
			13	15	16	13	14	16	13	14	15	B^S
			35	25	20	35	30	20	35	30	25	%

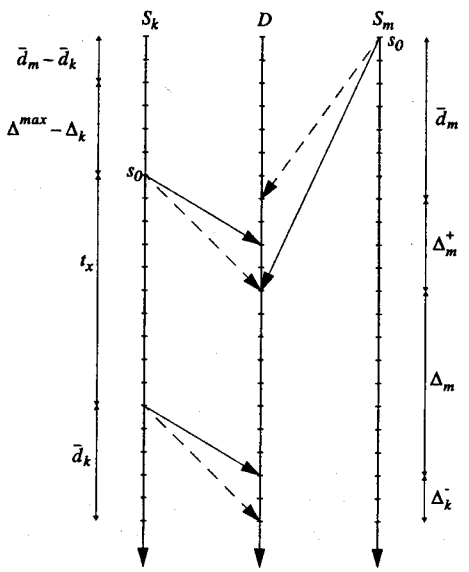

Fig. 20. Worst case scenario for different jitter values

because sequence numbers of the media units for the different synchronization groups are not related. We propose two techniques to cope with this problem. First, the greatest common divisor (GCD) of the media unit rates of all synchronization groups can be taken as a basis for determining the sequence numbers [Ran93]. This mostly implies a modification of the sequence numbers, after having captured a media stream. Second, a *mapping* of sequence numbers can be done. The synchronization group with the highest media unit rate serves as a reference. For instance, media unit number 6 of a group with a rate of 12 mu/s is mapped to media unit number 12 of a group with a rate of 24 mu/s. Depending on the media unit rates, this technique introduces inaccuracies by rounding.

5 Conclusion

We have presented a scheme for intra- and inter-stream synchronization of stored multimedia streams. Our scheme comprises three models that assure synchronization in an environment with different delays, jitter, server drop-outs, clock drift, and alteration of the average delay. The models do not rely on synchronized clocks within the network. In contrast to existing synchronization solutions, the scheme is suitable for streams that are striped across multiple server nodes, as well as for a single-server approach.

Model 1 presents a new protocol that allows to initiate the synchronized start-up of distributed multimedia streams. The protocol has little overhead, is easy to implement and is flexible with respect to different strategies of distributing the video material (e.g., shifted time order).

Model 2 provides intra- and inter-stream synchronization. Using a rule of Santoso et al., we formulate a condition that decreases the start-up latency of a stream without sacrificing smooth playout. Based on Santoso et al. and inspired by an idea of Ishibashi et al., we derive buffer requirements that assure intra- and inter-stream synchronization for multiple substreams originating from different sources. In contrast to Ishibashi et al., we could considerably reduce these requirements by applying our novel shifting strategy. Our experiments proved that models 1 and 2 obtain a perfect flow reconstitution in case of bounded jitter. Moreover, we provide new modified starting conditions for the smooth playout of a stream distributed across multiple sources.

Model 3 extends the previous two models by a buffer level control to cope with long-term effects such as clock drift. We introduce a new buffer model called virtual buffer that allows for faster reaction in response to disturbances. Our system model includes the source of a stream into the control loop, which allows to maintain the real-time continuity of a stream during playback. We investigate the trade-off between stability and reactivity when tuning the parameters of a filtering function and propose two strategies for restoring synchrony. Experimental results comparing these two strategies show the effectiveness of our buffer level control.

Models 1 to 3 have been successfully implemented in our video server prototype [Ber96], where each video is distributed (striped) over n server nodes.

Acknowledgements. The work described in this paper was supported by the Siemens-Nixdorf AG, Munich. We gratefully acknowledge the constructive comments of the anonymous reviewers.

Appendix

A Buffer savings

The values for B^M and B^S shown in Table 4 are rounded up based on a media unit rate of 25 mu/s.

B Shifting strategy with different delay values

In order to compensate for different delays when applying the start-up protocol, substream k is shifted $\bar{d}_m - \bar{d}_k$ seconds

forward on the time axis. According to the proof of Theorem 5, t_x is calculated as follows:

$$\begin{aligned} t_x &= \bar{d}_m + \Delta_m^+ + \Delta_m + \Delta_k^- - (\bar{d}_k + \Delta^{\max} - \Delta_k + \bar{d}_m - \bar{d}_k) \\ &= \Delta_m^+ + \Delta_m + \Delta_k^- - \Delta^{\max} + \Delta_k \\ &= \Delta_m^+ + \Delta_k^- + \Delta_k \\ &= \Delta_m^+ + 2\Delta_k^- + \Delta_k^+ \end{aligned}$$

References

[Aga94] Agarwal N, Son S (1994) Synchronization of Distributed Multimedia Data in an Application-Specific Mannerer. In: 2nd ACM International Conference on Multimedia, October 1994, San Francisco, Calif., ACM Press, pp 141–148

[Alm91] Almeida N, Cabral J. Alves A (1991) End-to-end Synchronization in Packet-Switched Networks. In: Herrtwich RG (ed) Second International Workshop on Network and Operating System Support for Digital Audio and Video, November 1991, Heidelberg, Germany (Lecture Notes in Computer Science vol. 614) Springer, Heidelberg, pp 84–93

[And91] Anderson DP, Homsy G (1991) A Continuous Media I/O Server and its Synchronization Mechanism. IEEE Comput 24(10) : 51–57

[Ber96] Bemhardt C, Biersack EW (1996) The Server Array: A Scalable Video Server Architecture. In: Effelsberg W, Danthine A, Ferarri D, Spaniol O (eds) High-Speed Networks for Multimedia Applications. Kluwer Academic Publishers, Dordrecht, pp 103–126

[Blu94] Blum C (1994) Synchronization of Live Continuous Media Streams. In: Proceedings of the 4th Open Workshop on High-Speed Networks, September 1994, Brest, France

[Bul91] Bultermann DC, Liere R van (1991) Multimedia Synchronization and UNIX. In: Herrtwich RG (ed) Network and Operating System Support for Digital Audio and Video, November 1991, Heidelberg, Germany (Lecture Notes in Computer Science, vol. 614). Springer, Berlin Heidelberg New York, pp 108–119

[Cen95] Cen S, Pu C, Staehli R, Cowan C, Walpole J (1995) A Distributed Real-Time MPEG Video Audio Player. In: Little TDC, Gusella R (eds) Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95), April 1995, Durham, N.H.(LNCS vol. 1018) Springer, Heidelberg, pp 142–153

[Cha94] Chakrabati A, Wang R (1994) Adaptive Control for Packet Video. In: Proceedings of the International Conference on Multimedia Computing and Systems, May 1994, Boston, Massachusetts, IEEE Computer Society Press, Los Alamitos, Calif., pp 56–62

[Eff93] Effelsberg W, Meyer T, Steimnetz R (1993) A Taxonomy on Multimedia Synchronization. In: Proceedings of the Fourth Workshop on Future Trends of Distributed Computing Systems, 1993, Lisbon, Portugal, IEEE Computer Society Press, Los Alamitos, Calif., pp 97–103

[Ehl94] Ehley L, Furht B, Ilyas M (1994) Evaluation of Multimedia Synchronization Techniques. In: International Conference on Multimedia Computing and Systems, May 1994, Boston, Mass., IEEE Computer Society Press, Los Alamitos, Calif., pp 514–519

[Esc94] Escobar J. Patridge C, Deutsch D (1994) Flow Synchronization Protocol. ACM Trans Networking 2(2): 111

[Geo96] Georganas ND, Steinmetz R, Nakagawa T (1996) Special Issue on Synchronization Issues in Multimedia Communications. IEEE J Sel Areas Conunun 14(1)

[Gey95] Geyer W (1995) Stream Synchronisation in a Scalable Video Server Array. Master's thesis. Institut Eurecom, Sophia Antipolis, France

[Ish95] Ishibashi Y, Tasaka S (1995) A Synchronization Mechanism for Continuous Media in Multimedia Conununications. In: IEEE Infocom'95, vol. 3, April 1995, Boston, Massachusetts, pp 1010–1019

[Eni95] Knightly EW, Wrege DE, Liebeherr J. Zhang H (1995) Fundamental Limits and Trade-offs of Providing Deterministic Guarantees to VBR Video Traffic. Performance Eval Rev 23: 98–107

[Koe94] Koehler D, Mueller H (1994) Multimedia Playout Synchronization Using Buffer Level Control. In: Steinmetz R (ed) 2nd International Workshop on Advanced Teleservices and High-Speed Connnunication Architectures, September 1994, Heidelberg, Germany. (LCNS 868), Springer, Heidelberg, pp 167–180

[Lit9O] Little T, Ghafoor A (1990) Synchronization and Storage Models for Multimedia Objects. IEEE J Sel Areas Commun 8(3): 413–427

[Lit91] Little T, Ghafoor A, Chang C, Berra P (1991) Multimedia Synchronization. IEEE Data Eng Bull 14(3): 26–35

[Lit92] Little TDC, Kao F (1992) An Intermediate Skew Control System for Multimedia Data Presentation. In: Rangan PV (ed) Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, November 1992, La Jolla, Calif. (LNCS 712), Springer, Heidelberg, pp 121–132

[Mas9O] Massalin H, Pu C (1990) Fine-Grain Adaptive Scheduling Using Feedback. Comput Syst 3(1): 139–173

[Mil91] Mills D (199 1) Internet Time Synchronization: The Network Protocol. IEEE Trans Commun 39(10): 1482–1493

[PL96] Perez-Luque MJ, Little TDC (1996) A Temporal Reference Framework for Multimedia Synchronization. IEEE J Sel Areas Commun 14(1): 36–51

[Ran93] Rangan PV, Ramanathan S, Vin HM, Kaeppner T (1993) Technique for Multimedia Synchronization in Network File Systems. Comput Commun 16(3): 168–176

[Rot95] Rothermel K, Helbig T (1995) An Adaptive Stream Synchronization Protocol. In: Little TDC, Gusella R (eds) 5th International Workshop on Network and Operating System Support for Digital Audio and Video, April 1995, Durham, N. H. (LNCS vol. 1018). Springer, Heidelberg, pp 178–189

[San93] Santoso H, Dairaine L, Fdida S, Horlait E (1993) Preserving Temporary Signature: A Way to Convey Time-Constrained Flows. In: IEEE Globecom, December 1993, Houton, Texas, pp 872–876

[Scr92] Screenan CJ (1992) Synchronisation Services for Digital Continuous Media. PhD thesis. University of Cambridge, Cambridge, UK

[Scr95] Screenan CJ (1995) Position Statement: To Use or Avoid Global Clocks. In: IEEE Workshop on Multimedia Synchronization (Sync'95), May 1995

[Ste9O] Steimnetz R (1990) Synchronization Properties in Multimedia Systems. IEEE J Sel Areas Commun 8(3): 401–412

[Ste95] Steinmetz R, Nahrstedt K (1995) Multimedia: Computing, Communications and Applications (Innovative Technology Series). Prentice Hall, Englewood Cliffs, N.J.

ERNST BIERSACK received his M.S. and Ph.D. degrees in Computer Science from the Technische Universität München, Munich, Germany. From March 1989 to February 1992 he was a Member of the Technical Staff with the Computer Communications Research District of Bell Communications Research, Morristown, USA. Since March 1992 he has been an Associate Professor in Telecommunications at Institut Eurecom, in Sophia Antipolis, France. His current research is on scalable reliable multicast transfer for very large groups, architectures for scalable high-performance video servers. Dr. Biersack is a member of the IEEE and ACM. He has been a program committee member for various international conferences and publicity chair for ACM SIGCOMM 97.

WERNER GEYER is Phd studen computer science at the Universit Mannheim, Germany. His research i ests include networked multimedia c puting, specifically, teleteaching, vi conferencing, video servers, mult applications. Werner received his in computer science and business ministration at Mannheim Unive in 1995. He is currently involve the TeleTeaching project Mannh Heidelberg which tries to estal computer-based synchronous dist learning between several universitie Germany.