In B. Butscher, E. Moeller, and H. Pusch, editors, *Interactive Distributed Multimedia Systems and Services (European Workshop IDMS'96, Berlin, Germany)*, volume 1045 of *LNCS*, pages 277–295. Springer Verlag, Heidelberg, Germany, March 1996

A Synchronization Scheme for Stored Multimedia Streams

Werner Geyer[1], Christoph Bernhardt, Ernst Biersack
Institut Eurécom[2]

Abstract: Multimedia streams such as audio and video impose tight temporal constraints due to their continuous nature. Often, different multimedia streams must be played out in a synchronized way. We present a scheme to ensure the continuous and synchronous playout of *stored* multimedia streams. We propose a protocol for the synchronized playback and we compute the buffer required to achieve both, the continuity within a single substream and the synchronization between related substreams. The scheme is very general because it only makes a single assumption, namely that the jitter is bounded.

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** [Ber95] 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). Between the information units of these streams a certain temporal relationship exists. Multimedia systems must maintain this relationship when storing, transmitting and presenting the data. Commonly, the process of maintaining the temporal order of one or several media streams is called **multimedia synchronization** [Eff93].

Continuous media are characterized by a well-defined temporal relationship between subsequent data units. Information is only conveyed when media quanta are presented continuously in time. As for video/audio the temporal relationship is dictated by the sampling rate. The problem of maintaining continuity within a single stream is referred to as **intra-stream** synchronization. Moreover, there exist temporal

[1] Now with: Praktische Informatik IV, University of Mannheim, 68131 Mannheim, Germany, geyer@pi4.informatik.uni-mannheim.de
[2] 2229 Route des Crêtes, 06904 Sophia-Antipolis — France, Phone: +33 93002611, FAX: +33 93002627, email: {bernhard,erbi}@eurecom.fr

relationships between media-units of related streams, for instance, an audio and video stream. The preservation of these temporal constraints is called **inter-stream** synchronization. To solve the problem of stream synchronization we have to regard both issues which are tightly coupled.

One can distinguish between **life synchronization** for life media streams and **synthetic synchronization** for stored media streams [Ste93a]. In the former case, the capturing and playback must be performed almost at the same time, while in the latter case, samples are recorded, stored and played back at a later point of time. For life synchronization, e.g. in teleconferencing, the tolerable end-to-end delay is in the order of a few hundred milliseconds only. Synthetic synchronization of recorded media stream is easier to achieve than life synchronization: higher end-to-end delays are tolerable, and the fact that sources can be influenced proves to be very advantageous as will be shown later. It is, for instance, possible to adjust playback speed or to schedule the start-up times of streams as needed. However, as resources are limited, it is desirable for both kinds of synchronization to keep the buffers required as small as possible. [Koe94]

1.3 Related Work

Escobar et al. [Esc94] and Rothermel et al. [Rot95b] 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, using time stamps. At the receiver different delays are equalized to the maximum delay by buffering. Rothermel enhances this basic mechanism with a *buffer level control* and a *master-slave* concept.

Rangan et al. [Ran93] present a synchronization technique based on feedback. Synchronization is done at the senders 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 to calculate the relative time deviation between sender and receiver. Agarval et al. [Aga94] adopt the idea of Rangan and enhance the scheme by dropping the assumption of bounded jitter.

Our synchronization scheme is inspired by the work of Santoso [San93] intra-stream synchronization and the work of Ishibashi et al. [Ish95] on intra-stream synchronization and inter-stream synchronization. Ishibashi proposes a time-stamp-based synchronization and applies a concept based on delay estimations to perform synchronization in case of unknown delay. Once intra-stream synchronization is established, inter-stream synchronization can be maintained with a certain probability. Corrective actions are taken by skipping/pausing. The scheme assumes no clock drift.

1.4 Context of the Synchronization Problem

The synchronization problem addressed in this paper is motivated by our work on scalable video servers. We have designed and implemented a video server, called **server array**, consisting of *n* **server nodes** . A video is distributed over all server

nodes using a technique called **sub-frame striping:** Each video frame is partitioned into *n* equal size parts, called **sub-frames**, that are stored on the n different servers. If $F_i = \{c_{i,1}, \ldots, c_{i,n}\}$ denotes the set of sub-frames for frame f_i, then: $\bigcup_{j=1,\ldots,n} c_{i,j} = f_i$

The server array with the synchronization mechanisms presented in this paper has been fully implemented as a prototype [Ber95].

During playback, each server node is continuously transmitting its (sub-frames) to the client. The transfer is scheduled so, that all striping blocks that are part of the same frame are completely received by the client at the deadline of the corresponding frame. The client reassembles the frame by combining the sub-frames from all server nodes. An example for *n=3* with each server sending with a rate of *r* frames per second is depicted in figure 1.

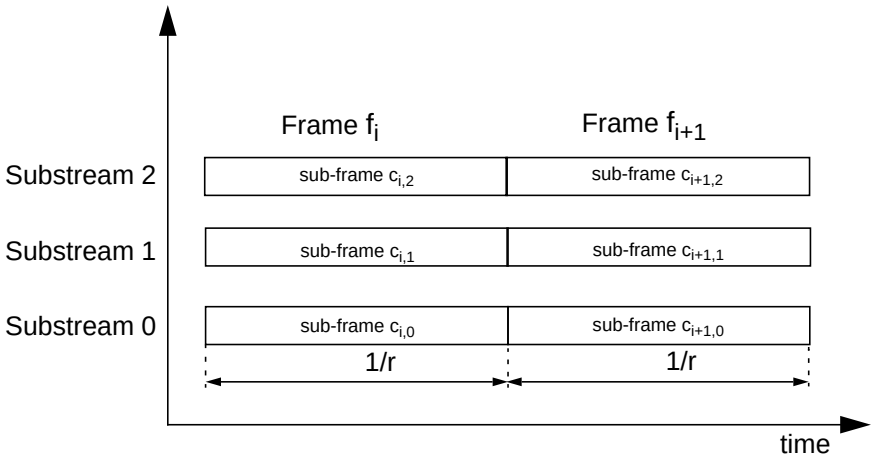

Fig. 1. Temporal Relationship for Sub-Frame Striping.

2 Synchronization Protocol

2.1 Overview

We propose a synchronization scheme for ***stored media*** that achieves both, suitable intra- and inter-stream synchronization. The scheme is *receiver-based* and does not assume global clocks. To initiate the playback of a stream in a synchronized manner we introduce a **start-up protocol**. Our protocol has been mainly influenced by the ideas of Ishibashi [Ish95] with respect to intra- and inter-stream synchronization. Based on Santoso's work [San93] we derive buffer requirements and playout deadlines to assure inter- and intra-stream synchronization. For *re*-synchronization, we adopt scheme similar to the one described by Koehler and Rothermel [Koe94], [Rot95c].

We derive our synchronization scheme by step-wise refinement: We first develop a solution for the case of zero jitter and then relax this assumption requiring bounded jitter only. We present two models:

Model 1 covers the problem of *different but fixed delays* on the network connections for each substream. We propose a 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 travelling from the source to the destination. Jitter is assumed to be bounded. To smoothen out jitter, elastic buffers are required. Our scheme guarantees a smooth playback of the stream and has very low buffer requirements. Model 2 covers intra-stream synchronization as well as inter-stream synchronization.

For the proposed synchronization scheme, we assume that a client **D** is receiving sub-streams from different servers[3]. Client and servers are interconnected via a network (see figure 2).

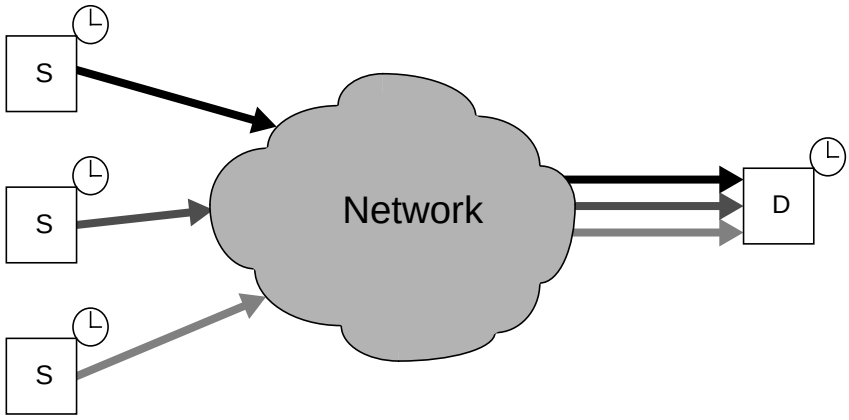

Fig. 2. Distributed Architecture for the Synchronization Scheme.

Each of the servers denoted by *S* delivers an independent **substream** of **media-units** (sometimes referred to as **frames** in the following). 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**, composed of the substreams, is driven by the destination's clock.

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

While the use of globally synchronized clocks facilitates synchronization, our synchronization scheme does *not* assume the presence of a global time or synchronized clocks.

2.2 Sources of Asynchrony

Several sources of asynchrony exist in the configuration described in the previous section. These are:

- **Different delays**: 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 for the reasons of *packet-size/depacketizing*, the processing through the lower protocol layers, and the buffering on the client site.
  The variation of delay is defined as **jitter**.
- **Network jitter**: asynchronous transfer destroys synchrony. Jitter arises in intermediate nodes for the reason of buffering.
- **End-system jitter**: packetizing and depacketizing of media-units with different size due to encoding introduces jitter as well as passing media-units through the lower protocol layers.
- **Clock drift** between the clocks in the servers and in the client is present because we do not assume global clocks.
- **Change of the average delay**: the synchronization scheme has to be adaptive with respect to a change of the average delay.
- **Server drop outs** due to process scheduling are a realistic assumption when using non-real-time operating systems. At the same time, the consideration of drop outs covers the overload probability of statistical admission control strategies to a certain amount.

2.3 Assumptions

Our synchronization mechanism uses **time stamps**. Each time a media-unit[4] is scheduled by a server, it is stamped with the current local time. This enables the client to calculate statistics, such as for the roundtrip delay, jitter, or inter-arrival times. Moreover, we assume that each media-unit carries a **sequence number** for determining 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

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

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 has to 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	$i, j, \upsilon = 0, ..., N\text{-}1$	
k	server index	$k = 0, ..., n\text{-}1$	
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		[sec]
s_i^c	synchronized sending time of media-unit i in server time		[sec]
a_i	arrival time of media-unit i in client time		[sec]
d_i	roundtrip delay[5] for media-unit i measured at client site		[sec]
d^{max}	maximum roundtrip delay		[sec]
$d^{max,j}$	maximum roundtrip delay for all j element of I_j		[sec]
t_{start}	starting time of the synchronization protocol		[sec]
t_{ref}	reference time for the start-up calculation		[sec]
t_{ref}^j	reference time regarding the set of media-units given by I_j		[sec]
t_i	expected arrival of the media-unit i at the client site		[sec]
δ_{ij}	arrival time difference between media-unit i and j		[sec]

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

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[6], i.e.

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

---

5 The roundtrip delay comprises the delay for a control message that requests a media-unit and the delay for delivering the media-unit

6 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 straight forward.

The leads to the following formulation of the **synchronization problem:**
The client must playout the media-units of all subsets I_j , with *j mod n = 0,* at the *same time.*

2.4 Model 1: Start-Up Synchronization

Introduction

Under the assumption of constant delay and zero jitter, we solve 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

$$t_i = t_0 \qquad \forall i \in I_0 \tag{2}$$

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 site [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 site 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 *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 afterwards automatically.

We need to introduce some more notation to express interdependencies 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 for the protocol.

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

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

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

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

Equation (5) computes the maximum of the roundtrip delay for all n substreams

$$d^{max} = max\{d_i | i \in I_0\} \tag{5}$$

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.

$$t_{ref} = max\{a_i | i \in I_0\} \tag{6}$$

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.

$$\delta_{ij} = a_i - a_j \quad \forall i, j \tag{7}$$

Start-Up Protocol

The synchronization protocol for starting playback on the server sites 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 $d_i \quad \forall i \in I_0$ for each connection, 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 Server 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 sends media-unit *i* time-stamped with s_i immediately 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 delay $d_i = a_i - t_{start} \quad \forall i \in I_0$ and
  the relative distance between media-unit arrivals $\delta_{ij} = a_i - a_j \quad \forall i, j \in I_0$ are computed.

(b) Synchronization Phase

- At local time t_{ref}, client D computes t_0 as $t_0 = max\{t_{ref} + d_i \mid i \in I_0\}$,
  the maximum round trip time as $d^{max} = max\{d_i | i \in I_0\}$,
  the index ν that determines t_0 as $\upsilon = \{j \in I_0 | t_{ref} + d_j = t_0\}$, and
  the delay differences as $\delta_{\nu i} = a_\nu - a_i$, $\forall i \in I_0$
- With these results the starting time of Server S_i is calculated in server time
  $s_i^c = s_i + d^{max} + \delta_{\upsilon i}$, $\forall i \in I_0$.
- Client D sends a *Sync_Request(i,* s_i^c*)* to server S_i, $\forall i \in I_0$.
- At local time $s_i + d_i + (t_{ref} - a_i)$ 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, $\forall i \in I_0$.
- 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 discuss the

- Calculation of t_0
- Calculation of s_i^c

(a) 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 given by (2). It is obvious that media-unit $i \in I_0$ delivered by server S_i can not be expected earlier than $t_{ref} + d_i$. Hence, the substream with the largest delay determines t_0.

Theorem 1: Let $t_0 = max\{t_{ref} + d_i \mid i \in I_0\}$.
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 $t_{ref} + d_i$ we need to show that $t_i \geq t_{ref} + d_i \quad \forall i \in I_0$.
Let $t_0 = max\{t_{ref} + d_i \mid i \in I_0\}$
$\Rightarrow t_0 \geq t_{ref} + d_i \quad \forall i \in I_0$
(2) $\Rightarrow t_i = t_0 \geq t_{ref} + d_i \quad \forall i \in I_0$
$\Rightarrow t_i \geq t_{ref} + d_i \quad \forall i \in I_0$
$\Rightarrow t_0$ does not violate the arrival times of other substreams
To show that t_0 is minimal, we assume that $\exists \quad \tilde{t}_0 < t_0$
$\Rightarrow \exists \ i_0 \in I_0$ with $\tilde{t}_0 < t_{ref} + d_{i_0}$
(2) $\Rightarrow \quad \tilde{t}_0 = t_{i_0} < t_{ref} + d_{i_0}$
$\Rightarrow$ 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 ν determining t_0 as follows:

$$\upsilon = \{j \in I_0 \mid t_{ref} + d_j = t_0\} \quad (8)$$

(b) Calculation of the Synchronized Sending Time s_i^c of Media-unit *i* for Server S_i

Substream υ can be considered *critical* since it determines the starting times of all other initial substreams. It 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 $\delta_{\upsilon i}$ between media-unit *i* and media-unit υ. This gives the starting time that provides a simultaneous arrival of the media-units of substream *i* and these of substream υ. The calculation based on (8), (7), (4) is stated in the following theorem.

Theorem 2: Let $s_i^c = s_i + d^{max} + \delta_{\upsilon i}, \ \forall\, i \in I_0$.
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 figure 3). If S_i sent media-unit *i* immediately back to *D*, it would arrive at client time $t_{ref} + d_i$. The term $s_i + d^{max}$ is corrected by $\delta_{\upsilon i}$.
Media-unit *i* will arrive at *D* at client time
$t_{ref} + d_i + \delta_{\upsilon i}$
$= t_{ref} + (a_i - t_{start}) + (a_\upsilon - a_i)$
(3) $= t_{ref} + a_\upsilon$
$= t_{ref} + d_\upsilon$
(theorem 1) $= t_0$
(2) $= t_i$ ■

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.

Example of the Start-Up Protocol

The following example in figure 3 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 $t_0 = max\{t_{ref} + d_i \mid i \in I_0\} = max\{23, 18, 24\} = 24$ we get $\upsilon = 2$.

Substream 2 experiences the longest roundtrip delay d^{max} and determines therefore 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.

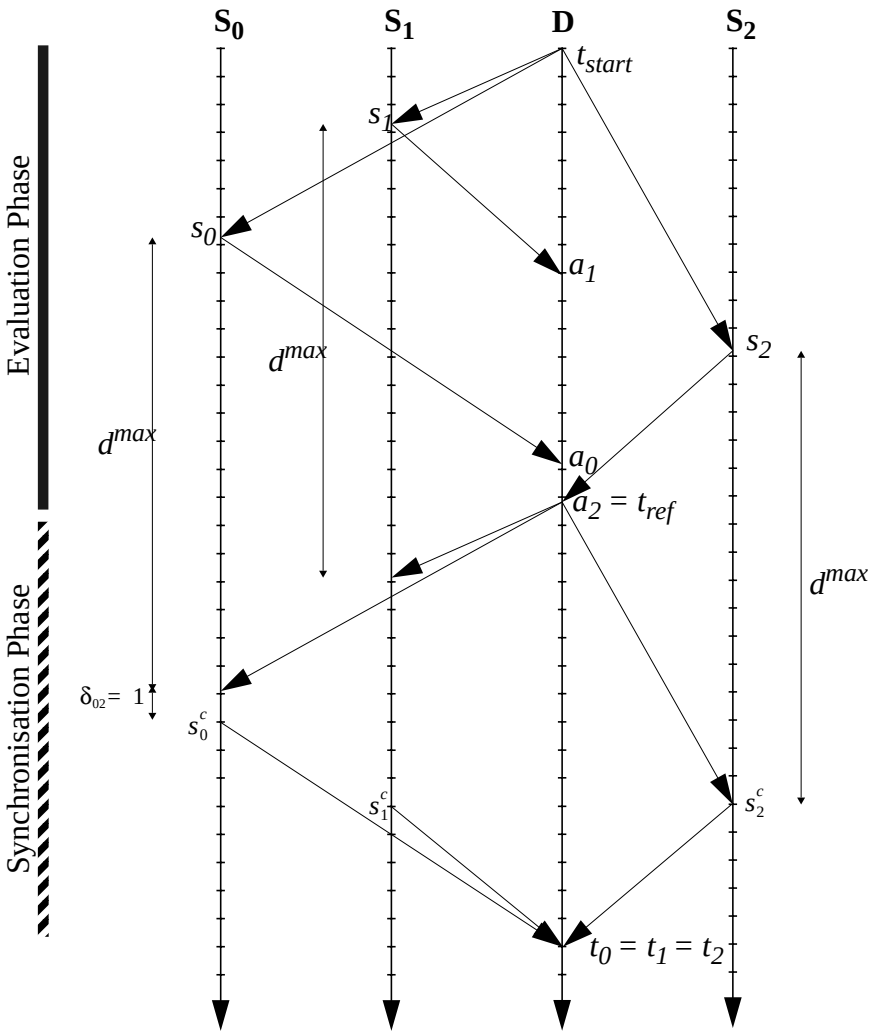

Fig. 3. Example of the Start-Up Synchronization Protocol Flow.

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	$s_2 + 12$

Table 1. Example for the Start-Up Calculations.

2.5 Model 2: Intra- and Inter-Stream Synchronization

Introduction

Model 1 shows how to cope with different delays for each substream. 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*. For our considerations, we regard the accumulated value of all sources of jitter described in section . Furthermore, 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 correct timely order. 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 smoothen out the effects of jitter, media-units have to be delayed at the sink such that a continuous playback can be guaranteed. Consequently *playout buffers* corresponding to the amount of jitter are required.

The main point addressed by model 2 is intra- and interstream synchronization and the calculation of the required buffer space. First, we regard the synchronization of a single substream. Based on a rule of Santoso [San93] we formulate a theorem that states a well defined playout time 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 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. Furthermore, the playout deadline is modified with respect to multiple substreams.

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 for the reason of 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

k	substream or server index,	$k = 0, \ldots, n\text{-}1$
r	requested display rate of each substream at client site	[mu/sec]

d_k^{max}	maximum delay for substream k	[sec]
d_k^{min}	minimum delay for substream k	[sec]
$\bar{d}_k$	average delay for substream k	[sec]
Δ_k	jitter for substream k	[sec]
Δ^{max}	maximum jitter of all substreams	[sec]
Δ_k^+	maximum upper deviation from $\bar{d}_k$ due to jitter for substream k	[sec]
Δ_k^-	maximum lower deviation from $\bar{d}_k$ due to jitter for substream k	[sec]
Δ^{max+}	maximum upper deviation of all substreams	[sec]
b_k	buffer requirement for substream k on sink site	[mu]
b_k^S	buffer requirement for substream k on sink site with shifting	[mu]
b_k^M	buffer requirement for substream k on sink site with max. jitter	[mu]
B	total buffer requirement for a synchronization group	[mu]
B^S	total buffer requirement for a synchronization groupwith 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[7].

$$\Delta_k = d_k^{max} - d_k^{min} \quad \forall k \tag{9}$$

$$\Delta^{max} = max\{\Delta_k | k \in \{0 \ldots n-1\}\} \tag{10}$$

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

$$\Delta_k = \Delta_k^+ + \Delta_k^- \quad \forall k \tag{11}$$

$$d_k^{max} = \bar{d}_k + \Delta_k^+ \quad \forall k \tag{12}$$

$$d_k^{min} = \bar{d}_k - \Delta_k^- \quad \forall k \tag{13}$$

$$\Delta^{max+} = max\{\Delta_k^+ | k \in \{0 \ldots n-1\}\} \tag{14}$$

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 sink to compensate the jitter. The buffer is emptied at a constant rate for displaying the media-units.

[7] Jitter is often defined as the *variation* of network delay.

Santoso [San93] has 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 be derived. The deadline given by Santoso (case a) can be lowered in some situations (case b).

Theorem 3: Smooth playout for a substream *k* can be guaranteed in case of bounded jitter whenever *either* one of the following two 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. Such a situation is shown in figure 4: the first media-unit experiences the maximum delay, subsequent media-units arrive in a burst (marked gray in figure (4)) 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 dotted lines.
Assuming that the $(\lceil \Delta_k \cdot r \rceil + 1)$*-th* media-unit just has 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 is $\Delta_k^- + r^{-1} + \Delta_k^+ = \Delta_k + r^{-1}$ seconds. $(\lceil \Delta_k \cdot r \rceil + 1) \cdot r^{-1}$ gives an upper bound for $\Delta_k + r^{-1}$. 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 required minimum buffer space for the synchronization of a single substream.

Theorem 4: To guarantee intra-stream synchronization for a single substream by applying theorem 3, a minimum buffer space of $\lceil 2\Delta_k \cdot r \rceil$ media-units is required.

For a proof see [Gey95].

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 the latter 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 experiences the *same jitter*. In the following, we consider and examine the impact on buffer requirements for different jitter bounds for each substream . This reflects the

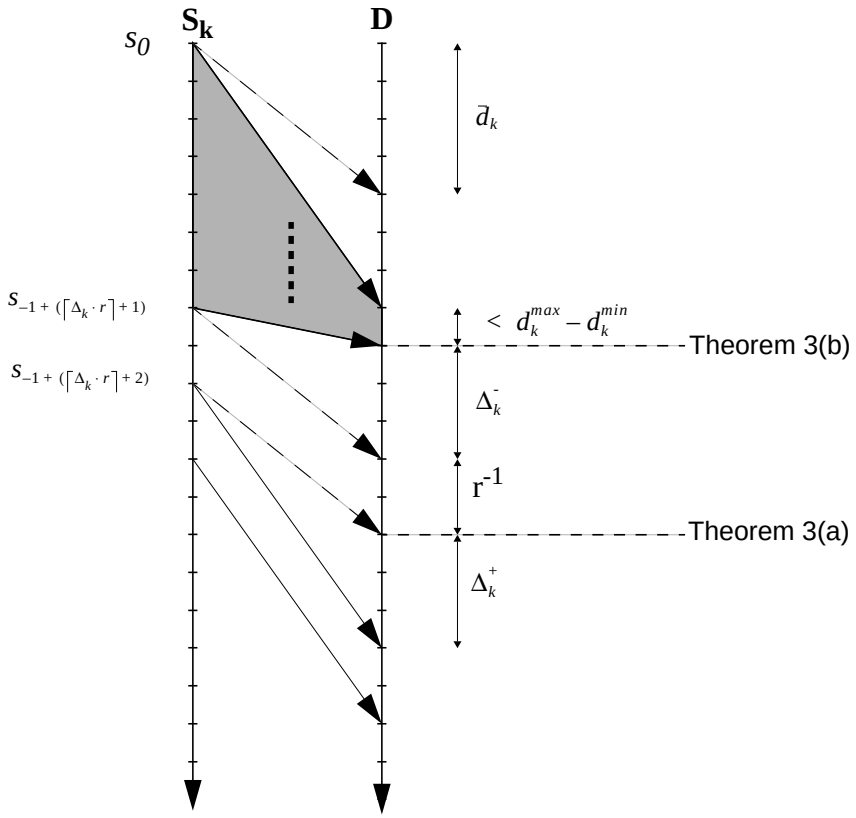

Fig. 4. Worst Case Scenario for a Single Substream.

case that the paths from sources to the destination are independent. We assume that media-units experience an average delay $\bar{d}$ on all substream connections. The following proofs can also carried out with different delays.

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 given by a *synchronization group* is defined by the latest substream that satisfies theorem 3. 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. To cope with this problem in a straight forward manner, Ishibashi et al. [Ish95] propose to allocate the buffer according to 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.

$$b_k^M = \lceil 2\Delta^{max} \cdot r \rceil \qquad (15)$$

$$B^M = \sum_{k=0}^{n-1} b_k^M = n \cdot \lceil 2\Delta^{max} \cdot r \rceil \qquad (16)$$

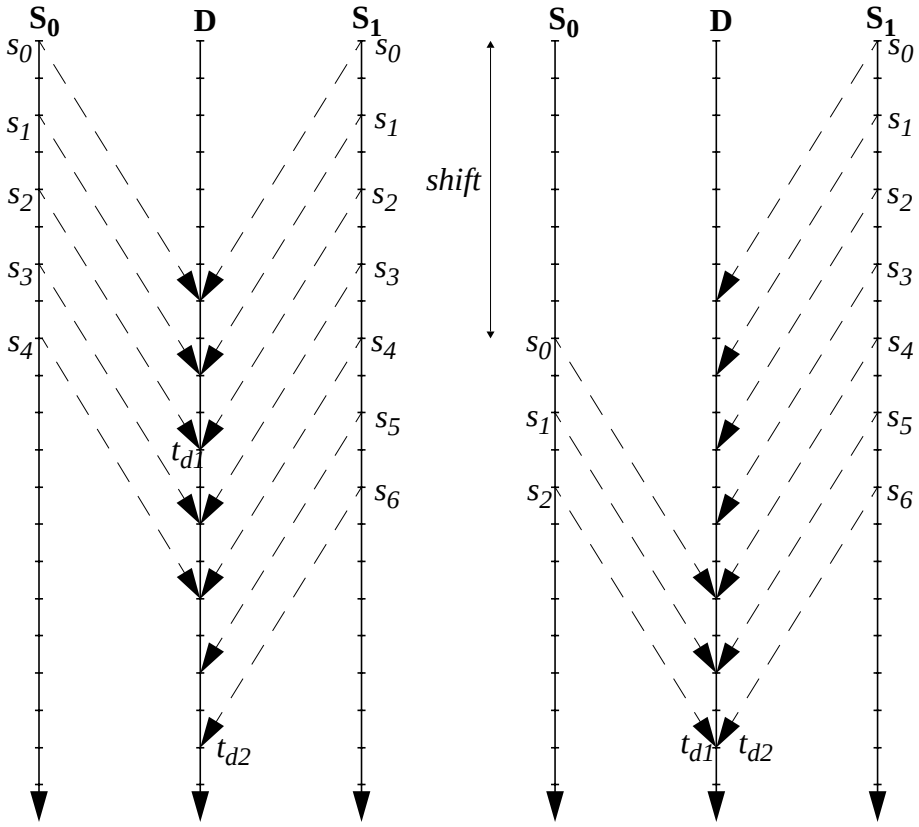

Fig. 5. Multiple Substreams without and with Shifting.

(b) Shifting Strategy

Depending on the differences in the jitter values for 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 on average *at the same time*. This is done by shifting the starting points of all substreams according to the deadline of the substream, with the largest jitter bound. Figure 5 depicts such a scenario for two sources[8] 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 1 and twelve media-units for substream 2. Substream 1 reaches its playout deadline on average at t_{d1} and substream 2 at t_{d2}. Without shifting a buffer overflow occurs when receiving the 5th media-unit of substream 1 while substream 2 still has to wait two time units until playout can commence. By shifting, both substreams arrive 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 time axis with the difference of its jitter to the maximum jitter, i.e. $\Delta^{max} - \Delta_k$ seconds. Clearly, substream k has to be started $\Delta^{max} - \Delta_k$ seconds later than the substream with the highest 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 exists a worst case that requires additional buffer space for each substream. The amount of additional buffering is stated in theorem 5.

Theorem 5: Applying a shift of $\Delta^{max} - \Delta_k$ to the *k-th* substream, $k = 0, \ldots, n\text{-}1$, and having bounded jitter for each substream, inter-stream synchronization for multiple dependent substreams can be guaranteed if in addition to the buffer requirement of theorem 4, another $\lceil (\Delta^{max+} - \Delta_k^+) \cdot r \rceil$ buffer slots are allocated.

For a proof see [Gey95].

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

$$b_k^S \underset{n-1}{=} \lceil (2 \cdot \Delta_k + \Delta^{max+} - \Delta_k^+) \cdot r \rceil \tag{17}$$

$$B^S = \sum_{k=0} b_k^S \underset{n-1}{=} \sum_{k=0} \lceil (2 \cdot \Delta_k + \Delta^{max+} - \Delta_k^+) \cdot r \rceil \tag{18}$$

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, 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 this delay corresponds to the average delay, 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. Since the request message is a small packet made up of sev-

[8] In contrast to the definitions for model 1, each one of the depicted substreams delivers equal media-unit numbers.

eral bytes the following considerations we will neglect the jitter experienced by the request packet, supposing that the jitter bounds for the buffer calculations stated in theorem 4 and 5 have been chosen sufficiently large.

2.7 Exact Buffer Requirements

Model 2 gives us 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 6. Throughout all theorems, we expressed the time we need to buffer in terms of media-units. The required buffer space can be optimized by summing up the time to buffer given by theorem 4 and 5 and by transforming the resulting sum into buffer slots. We can summarize the overall buffer requirements b_k for a substream k and B for a synchronization group consisting of n substreams as follows.

$$b_k = \left\lceil \left(2\Delta_k + (\Delta^{max+} - \Delta_k^+) + max\{\Delta_m + \Delta_k^+ - \Delta^{max+} \mid m \neq k \wedge m = 0 \ldots n-1\} \right) \cdot r \right\rceil \tag{19}$$

$$B = \sum_{k=1}^{n} b_k =$$

$$\sum_{k=1}^{n} \left\lceil \left(2\Delta_k + \Delta^{max+} - \Delta_k^+ + max\{\Delta_m + \Delta_k^+ - \Delta^{max+} \mid m \neq k \wedge m = 0 \ldots n-1\} \right) \cdot r \right\rceil \tag{20}$$

3 Conclusion

We have presented a scheme for intra- and inter-stream synchronization of stored multimedia streams. The only assumption we make is that jitter is bounded, which is typically true in todays networks. Having presented a base-version of the synchronization scheme, we make enhancements such as shifting the start-up times in order to reduce the buffer requirements. At the end, we derived exact bounds for the buffer requirements.

Acknowledgment

The work described in this paper was supported by the Siemens Nixdorf AG, Munich.

4 References

[Aga94] N. Agarwal and S. Son. “Synchronization of Distributed Multimedia Data in an Application-specific Manner.” In *2nd ACM International Conference on Multimedia*,

pages 141–148, San Francisco, USA, October 1994.

[Ber95] C. Bernhardt and E. Biersack. “The Server Array: A Novel Architecture for a Scalable Video Server.” In *Proceedings of the Distributed Multimedia Conference*, pages 63–72, Stanford, USA, August 1995.

[Eff93] W. Effelsberg, T. Meyer, and R. Steinmetz. “A Taxonomy on Multimedia-Synchronization.” In *Proceedings of the Fourth Workshop on Future Trends of Distributed Computing Systems, Lisbon, Portugal, Sep. 1993*, pages 97–103. Eyrolles, 1993.

[Esc94] J. Escobar, C. Patridge, and D. Deutsch. “Flow Synchronization Protocol.” In *ACM Transactions on Networking*, volume 2, pages 111–121. IEEE, April 1994.

[Gey95] W. Geyer. “Stream Synchronisation in a Scalable Video Server Array.” Master’s thesis, Institut Eurecom, Sophia Antipolis, France, September 1995.

[Ish95] Y. Ishibashi and S. Tasaka. “A Synchronization Mechanism for Continuous Media in Multimedia Communications.” In *IEEE Infocom’95*, volume 3, pages 1010–1019, Boston, Massachusetts, April 1995.

[Koe94] D. Koehler and H. Mueller. “Multimedia Playout Synchronization Using Buffer Level Control.” In *2nd International Workshop on Advanced Teleservices and High-Speed Communication Architectures*, pages 165–180, Heidelberg, Germany, September 1994.

[Ran92] P. V. Rangan, H. M. Vin, and S. Ramanathan. “Designing an On-Demand Multimedia Service.” *IEEE Communications Magazine*, 30(7):56–65, July 1992.

[Ran93] P. Rangan, S. Ramanathan, H. M. Vin, and T. Kaeppner. “Techniques for Multimedia Synchronization in Network File Systems.” *Computer Communications*, 16(3):168–176, March 1993.

[Rot95b] K. Rothermel and T. Helbig. “An Adaptive Stream Synchronization Protocol.” In *5th International Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, New Hampshire, USA, April 1995.

[Rot95c] K. Rothermel, T. Helbig, and S. Noureddine. “Activation Set: An Abstraction for Accessing Periodic Data Streams.” In *Multimedia Computing and Networking*, volume 2417, San Jose, California, February 1995. IS&T/SPIE.

[San93] H. Santoso, L. Dairaine, S. Fdida, and E. Horlait. “Preserving Temporal Signature: A Way to Convey Time Constrained Flows.” In *IEEE Globecom*, pages 872 – 876, December 1993.

[Ste93a] R. Steinmetz. *Multimedia-Technologie*. Springer Verlag, Heidelberg, Germany, 1993.