Synchronization issues in distributed applications: definitions, problems, and quality of synchronization

Synchronization Issues in Distributed Applications: Definitions, Problems, and Quality of Synchronization

Christina Class

December 5, 1997

Abstract

In this report synchronization issues arising in distributed applications are discussed.

A general and detailed introduction to the field of synchronization is provided. General terms related to synchronization as well as concepts are introduced, defined and discussed in short. The specific problem fields of synchronization in multi-party environments and in presence of control commands are introduced. A discussion of some of the numerous related work dealing with different aspects of synchronization completes this introduction to synchronization.

A system model is introduced that describes a communication system infrastructure being able to guarantee for end-to-end Quality of Service. The system model provides a precise description and definition of the environment and allows for the abstraction from technical details. The synchronization study of this report is based on the system model.

Intra- as well as inter-stream synchronization is defined in a formal manner and their establishment and maintenance are described. Special attention is paid on distributed applications with multi-party connections as well as on control commands influencing the course of the presentation. Both issues are critical to synchronization and often not treated adequately in published work.

A notion of synchronization quality is developed and defined. The notion comprises the maximum remaining asynchrony, delays, error probabilities and buffer requirements. Based on the given environment and synchronization method values for the parameters of the synchronization quality are calculated.

This report does not serve as a guideline for a new synchronization scheme, but presents relevant topics in the area of synchronization, provides formal definitions of synchronization and investigates, in an optimal and theoretical environment, the criteria for synchronization quality.

Contents

1 Introduction

Multimedia applications are characterized by their integrated processing of different media types such as text, video, audio. Some of these media types are time-dependent, i.e. their value changes in function of time. The recording, storing and presentation of these media must take into account the time component of the data. The task of multimedia synchronization is to retain these temporal relationships. Distributed multimedia applications became possible as the available bandwidth in computer networks increased and data transmission became more reliable. The distribution of multimedia data adds delay that may vary over time. Data may get lost or corrupted during transmission. Depending on the underlying network infrastructure even the sequence of packets in packet switching data networks is not necessarily retained. These factors complicate the task of multimedia synchronization in distributed applications.

The work presented discusses issues of synchronization in distributed multimedia applications. Synchronization is discussed in an abstract way with the main focus on three topics:

- synchronization in multi-party applications
- synchronization in applications providing interactive control commands to the user
- quality of synchronization

In chapter 2 relevant terms for synchronization are defined. The basic synchronization concepts, temporal, event-based and content-based synchronization, are presented as well as the basic synchronization methods: time-stamping, interleaving and out-of-band synchronization. Basic synchronization issues related to multicasting and to control commands are also introduced.

After the introduction of basic terms and concepts, chapter 3 provides an overview of related work.

To discuss synchronization issues in distributed applications a model of the technical environment has to be presented as a basis for discussion, even if the abstraction level is high. Our system model is introduced in chapter 4. This model allows for the abstraction of the concrete technical environment as a computer system, communication protocol, and underlying network infrastructure to enable the examination of synchronization issues in distributed applications on an abstract, theoretical level. The system model is flow based, i.e. one data stream is referred to as a reference unit for data transmission. The system model allows for the specification of flow based application requirements. The synchronization issues as discussed in chapter 5 through chapter 7 assume that these requirements are guaranteed.

Chapter 5 introduces intra-stream synchronization. The synchronization method is presented in detail for a point-to-point communication scenario with constant delay. Subsequently, this assumption is dropped and the method is presented for the case of variable delay as well as for the general case. The topic of control commands, multipoint-to-point, point-to-multipoint, and multipoint-to-multipoint communication as well as the assumptions on the system environment that must hold for the synchronization method to be implemented are treated in one subsection each.

In chapter 6, our inter-stream synchronization method is presented. After defining and discussing inter-stream synchronization in general, the method is presented for the multipoint-to-point scenario. In the second part of the chapter, control commands, point-to-point, point-to-multipoint, and multipoint-to-multipoint communication as well as our assumptions are discussed. In this chapter the problem of synchronization in presence of control commands and multicasting is not yet entirely solved.

In chapter 7 the presented method is analyzed and a basic notion of *synchronization quality* is developed. The maximum asynchrony that can be guaranteed by the method as well as synchronization error probabilities, buffer requirements and delays added by the method are calculated. These values may serve as a basis to compute synchronization quality.

Chapter 8 reasons about the validity of the discussion by presenting a possible implementation of the system model as well as of the synchronization method. The indications are still not very detailed due to the theoretical focus of this report.

The report is concluded with a short summary and lists some future work in chapter 9.

2 Synchronization of Multimedia Data

In this chapter the terminology as used in this report is introduced by defining the relevant terms for synchronization. Using this terminology, basic concepts for synchronization in (distributed) multimedia systems are described and different kinds of synchronization errors are defined. Two specific problem fields are introduced and motivated: synchronization in multi-party applications and in applications allowing for control commands.

2.1 Definitions

As data is stored and processed on computers in discrete values (bits and bytes), *continuous* data streams like sound, audio, and light, video, have to be digitized and quantized. One *presentation unit (PU)* contains the atomic information of a media stream that can be displayed. This is, e.g., an audio sample or a video frame. The duration of one PU in data streams is media dependent and can vary a lot. In this work we assume that the PUs are *independent* from each other and can be processed independently.

Video compression methods exist as MPEG, that achieve compression by only coding the differences between given video frames or by using a frame prediction based on e.g. motion vectors and coding the difference between predicted and real frame, only. These schemes introduce dependencies between different coded PUs. When the PUs are uncompressed they are again independent from each other to be displayed/processed. This work is based on the assumption of independent PUs. For this reason, either compression schemes are to be used that do not add any interdependencies between PUs or the PUs must be compressed in the sender after the synchronization relevant task is performed and decompressed in the receiver before the synchronization relevant task is performed. The use of a compression method creating dependencies between different information units has an influence on the overall bit error probability. If a packet containing one information unit other PUs are depending of is corrupted, the other information units may be false or worthless, too.

This document focuses on synchronization of multimedia data. According to [Ste90] *multimedia* is the "integrated generation, presentation, processing, storage, and dissemination of independent machine processable information expressed by means of time dependent and time independent data".

Time independent data or *discrete media* does not have a time component. The validity and, thus, correctness of the data does not depend on any time conditions [SN95]. Examples for time-independent media are text, isolated pictures, graphics, etc.

The value of *time dependent data* changes over time, and its validity depends on time conditions, i.e. there might exist time intervals in which the data simply is not valid at all. For this reason, data needs not to be processed if its *valid time interval* is elapsed or has not already started. The valid time interval refers to the start of the presentation of one PU. If a PU corresponds to an audio sample of the length of $125\mu s$, a valid time interval of $[x, y]$ means that a valid presentation of the PU starts within $[x, y]$ and ends within $[x + 125\mu s, y + 125\mu s]$. The presentation duration of this single PU of $125\mu s$ does not influence the valid time interval and is not influenced by it. Time dependent data can be referred to as *data streams*.

Time independent data becomes time dependent within a given presentation if it has to be processed/displayed together with a specified time dependent presentation unit or if it has to be displayed at a specified point within a specified data stream.

In this report we use the following temporal definition of the term synchronization.

Definition The task of *synchronization* of multimedia data is to guarantee that all time dependent presentation units are only presented within their valid time interval. The valid time interval for each presentation unit is specified within the *synchronization specification* of multimedia data.

As synchronization is achieved by assuring the in time display of presentation units, *time* plays an important role for synchronization. Time is measured inside the computers using *clocks*. These crystal-based clocks are subject to *clock drift*, i.e. they count the time at different rates [CDK94]. Clock drift

causes the time in different computer systems to be different. In this report we assume *no clock drift*. This cannot be exactly achieved and thus a *bounded clock drift* is required[1]. With clock drift we only refer to the advancement in time at different speeds in different clocks. This assumption is essential for the definition of intra- and inter-stream synchronization in chapter 5 and 6.

2.2 General Synchronization Mechanisms

Synchronizing multimedia data can be divided into two different tasks. *Intra-stream synchronization* guarantees that data of one media stream is only presented/processed within its valid time interval. As different independent media can be integrated within a multimedia system, relations between the data of these different media must be defined. If at least one of these data is time dependent, the relations themselves have a time dependent validity. Data that is related has to be presented/processed together, i.e. within a specified time interval. This is the task of *inter-stream synchronization*.

Synchronization issues are widely discussed in the literature (see chapter 3). There are different approaches how to design synchronization. We can distinguish the *content-based*, *event-based* and the *temporal synchronization* approach.

The *content-based synchronization* approach assigns a content to each PU and defines relations between different PUs of a given content. The relations between the different PUs have to be specified within a *composition schedule*. Nevertheless, time information is needed to specify the duration of the presentation of PUs. This approach is not further examined within this paper.

The *event-based synchronization* method can be projected to both, content-based and temporal synchronization. The event-based synchronization assumes that the processing of PUs is controlled by events. Events can be triggered by timer objects. This results in the temporal synchronization approach. Events also can be triggered by other PU objects. This corresponds to a relation between the two PUs as described in a composition schedule.

In the *temporal synchronization* approach, the valid time interval is assigned to each PU. This approach corresponds to our definition of synchronization on page 5. The synchronization algorithm guarantees that the PU is processed if available within its valid time interval or not processed at all. To achieve this, a time point t_P is assigned to each PU. The valid time interval is defined as $[t_P - x, t_P + y]$, where $\Delta = x + y$ defines the maximum accepted asynchrony for the PU. To do so, we need a time monitoring process that invokes the processing of a PU within its valid time interval. This can be realized using timers that trigger an alarm at the beginning of the valid time interval. The time point t_P assigned to each PU can be of different types:

- *Registration time:* this time can be used for preorchestrated data and for synthetic data. The time point's value is the time when the PU was captured/created. Relations between different data streams can only be kept if the streams are captured/created at the same time.
- *Sending time:* the time point specifies the time the PU was captured/read within the distributed application to be sent to the receiver. Relations between different streams must be restored before sending the data, thus that data that has to be presented/processed at the same time is sent at the same time. Relations between different data streams can only be kept if the streams originate from the same sender or if the receiver can map from the time in one sender to the time in other senders.
- *Reference time:* the time point specifies the time the PU has to be displayed/processed within the application. This time is specified as a relative value. Reference time zero denotes the time the first PU of a data stream that is valid of the beginning of an application is to be presented/processed. All related data streams can be captured and sent independently if a common zero time point is defined.

In case the reference time is assigned to PUs, the time points do not necessarily be assigned directly to each PU. They also can be stored independently from the PUs in a media schedule provided that each

[1] The bound in clock drift adds as additional asynchrony.

PU can be unambiguously identified. From sending or registration time the reference time can easily be obtained by simply subtracting the time of the first PU from the time of the PUs. Thus, the first PU in the application gets 0 assigned to as reference value and all other PUs get the time difference from the first PU to itself assigned to as reference time.

A *media schedule* is an abstract description of the valid time intervals of PUs and always starts with a 0 time point. A *presentation schedule* is the mapping of a media schedule to a real world application. The 0 time point of the media schedule is mapped to the start point of the applications. This time offset is then added to all specified times in the media schedule to obtain the presentation schedule that is only valid for this specific application at this specific place.

Synchronization is performed using the reference, the sending or the registration time. This time may be assigned to each PU to properly perform synchronization. The assignment of a registration, sending or reference time to a PU is referred to as *time-stamping*, the time assigned to the PU is the *time stamp*.

Time dependent data can be *periodic* or *non periodic*. Data is periodic if the PUs describing the data have to be presented/processed periodically. The *period* is the number of PUs that has to be presented with constant time differences per second. Let ω denote the period.[2] Then the PUs have to be presented/processed at time $t_0 + i \cdot 1/\omega$, where i denotes the number of the PU. The time stamp of PUs in case of periodic data is implicitly given by the period ω and the number i of the PUs. In case of non periodic data, the time stamp has to be explicitly obtained.

Intra-stream synchronization involves two aspects, the timing as well as the ordering aspect. The *timing aspect* refers to the synchronized timed display of a presentation unit. One presentation unit is only valid at a given point of time within the data stream and has to be displayed as long as it was recorded/captured. The *ordering aspect* of synchronization refers to the order of the presentation units. These have to be displayed in the same order as they had been captured or as specified in the media schedule, respectively. If the timing aspect of synchronization is fulfilled the ordering aspect of synchronization also is fulfilled but not vice versa. The complexity of synchronization algorithms may be decreased if these two aspects are treated independently.

A *distributed multimedia application* is defined as a multimedia application where at least one time dependent data stream is transmitted via a communication link to the presentation end system. In distributed multimedia applications not only the data i.e. the PUs but also the synchronization information has to be transmitted from the sender to the receiver. To do so, three different possibilities exist (see also [SSF95]):

- *Interleaving* or *multiplexing*: this methods can be applied in case of inter-stream synchronization. Data streams that have to be synchronized are transmitted over the same communication channel. PUs that have to be displayed together are transmitted together. Interleaving of PUs is shown in figure 1. This method cannot be applied if the different data streams come from different sources. Some additional information must be transmitted to correctly perform the intra-stream synchronization. To do so, one of the following two methods can be applied.

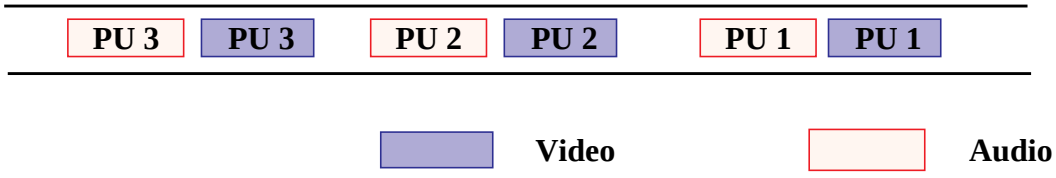

Figure 1: Interleaving of PUs

- *Out-of band synchronization information*: instead of one communication channel transporting the data, two channels are used. One channel transmits the PUs, the second channel is used to transmit the synchronization information (see figure 2). This information is thus transmitted as out-

[2] Appendix A summarizes the mathematical symbols used within this report.

of-band information. The synchronization channel, i.e. the channel transmitting the synchronization information, should be a reliable channel so that the synchronization information is available to the receiver when it receives the PUs or at least when the received PUs are to be presented.

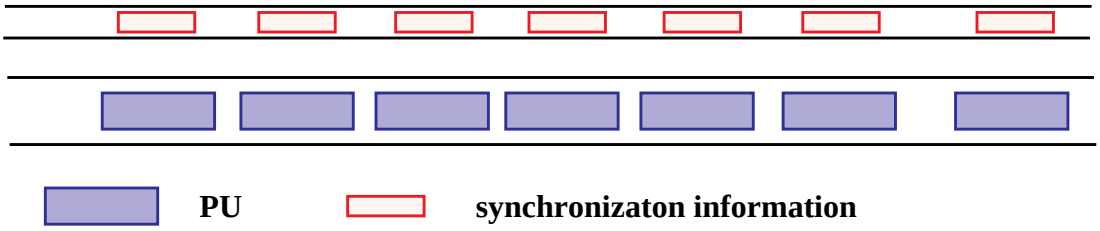

Figure 2: Out-of-band Synchronization Information

- *Time-stamping*: the timing information is assigned to each PU and transmitted together with the PU. The synchronization information is thus transmitted as in-band information (see figure 3).

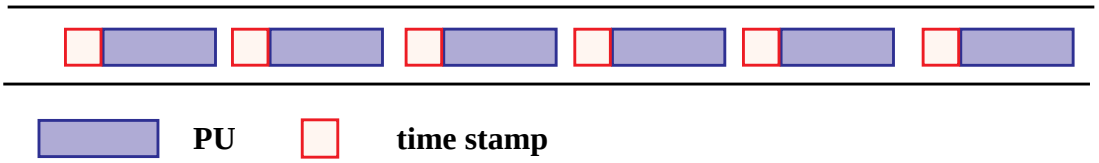

Figure 3: Time-stamping

2.3 Error Cases in Synchronization

Different conditions result in situations when the presented data streams on the receiver's site do not exactly correspond to the data streams as they have been created or captured. That is, at least one presentation unit is not displayed within its valid time interval. This is a *synchronization error*. Synchronization errors can be of three different types.

2.3.1 Gaps in Data Streams

A *gap* in a data stream can be defined as one or more consecutive valid time intervals of PUs in which the corresponding PU is not displayed. A gap can have multiple reasons:

- Late arrivals of PUs: a PU arrives late if it is at the data output point after the valid time interval has passed by. Thus, the PU cannot be presented/processed within its valid time interval.
- Loss of PUs: PUs that contain bit errors[3] are not delivered to the application. They are lost. These PUs cannot be presented/processed.
- Unreliable system: the environment of the receiver may be unreliable leading to a gap in a data stream.

The gap in this paragraph is an erroneous situation and results from a misbehavior in the network connection or on the end-systems and is different from the gap as defined in [Ste90]. [Ste90] elaborates on the gap problem in inter-stream synchronization. This problem occurs e.g. if a video sequence is longer than the audio it is to be synchronized to.

[3] In case of a Forward Error Correction scheme, the bit error rate is decreased.

2.3.2 Loss of Synchronization

A *loss of synchronization* is defined as a PU that is displayed outside its valid time interval. A loss of synchronization may result from

- unperceived bit errors that modify the synchronization information
- delay in the end-system after the synchronized play out has been triggered
- errors in the synchronization module

Loss of synchronization results in noise.

2.3.3 Loss of Time

Every temporal synchronization method requires a mapping between the time system of the sending/capturing end-system and the time system of the receiving/presenting end-system. Even if both, capturing and presenting, is done in the same end-system, this mapping is required if both are not performed simultaneously. If no mapping is done or a mapping is performed that is not valid, there is a *loss of time*.

A loss of time may be caused by drifting clocks. Clock synchronization is a well-known problem in distributed systems. [Lam78] describes the orders of events in distributed systems and the computation of logical as well as of physical clocks. A proof is also provided that the clock drift of the calculated physical clocks is bounded. There are protocols like the *Network Time Protocol (NTP)* in the Internet [Mil91] allowing for synchronizing clocks. With NTP the time accuracy can mostly be maintained to within a few milliseconds even in most parts of the Internet [Mil91]. By such protocols, the clock drift can be bounded, leading to a higher asynchrony but avoiding the situation of loss of time.

All mentioned error cases can occur within real systems. The probability of synchronization errors in a given environment for a given presentation is a function of the error probabilities of all three possible error types mentioned above for the given environment and the given presentation. In chapter 7.2 we elaborate on the error probability for the synchronization method as presented in chapter 5 and 6 in the given environment of our system model presented in chapter 4.

2.4 Multicasting

Many multimedia application like Video on Demand have multiple receivers. *Multicasting* allows for transmitting the data from the sender to receivers using a *single, atomic* operation. Video Conferencing is an application with not only multiple receivers but also multiple senders. In both applications audio and video are to be transmitted and to be synchronized on the receiver's site. Due to this reason, synchronization issues are relevant also in applications using multicasting.

[Bau97] defines a *multicast group* as the "set of receivers that is reached by a multicast". "In *point-to-multipoint multicasting*, a single sender distributes data to a multicast group. In *multipoint-to-multipoint multicasting* several senders send data to a multicast group. Membership changes in the multicast group affect all senders. In both cases [point-to-multipoint multicasting and multipoint-to-multipoint multicasting], the sender(s) may be receivers at the same time."[Bau97]

Applications can support connectivity between one or more senders and one or more receivers. This results in the following four different scenarios.

Point-to-point The case of a point-to-point connection[4] is the most simple case. A point-to-point connection is a single connection/communication link between one sender and one receiver. The point-to-point connection type is the easiest in what concerns intra-stream as well as inter-stream synchronization as only two different time systems are relevant for synchronization.

[4]The term connection in this case refers to a logical connection. In case of multipoint-to-multipoint this "connection" is implemented by establishing multiple physical connections between each one sender and the multicast group.

Point-to-multipoint A point-to-multipoint connection is the classical case of multicasting. Data is sent from one sender and distributed to multiple receivers. The sender sends the data to a multicast address. This address specifies a multicast group. Each receiver is part of this group and automatically receives all sent data as long as it remains in this group. Multicasting often allows for dynamically joining and leaving to/from the multicast group.

Multipoint-to-point A multipoint-to-point connection is a connection between multiple senders and one receiver. Such connections are implemented in the receiver by specifying point-to-point connections for data flows for every sender. An example for such connections may be audio and video of a movie that is coming from different servers (an audio and a video server, e.g.) and has to be presented in a synchronized matter at the receiver's site. Multipoint-to-point is especially interesting for inter-stream synchronization as it is very likely that connections to different senders suffer different delays and different jitters depending on the topology of the underlying network as well as on distances between receiver and the different senders.

Multipoint-to-multipoint If multiple senders are sending data that is received by multiple receivers we speak of multipoint-to-multipoint connections. This is the most crucial scenario concerning synchronization, as the sender group as well as the receiver group has to be synchronized to enable the synchronized data transmission and presentation. Multipoint-to-multipoint connections can be established when every sender sends data to a multicast group and when every receiver establishes flows to connect to all senders.

If a receiver performs a dynamic join to a multicast group that transmits synchronized data, information has to be transmitted to this new receiver so that it can perform the synchronization itself. It remains subject to further investigation which information is necessary for the receiver to support synchronization after a dynamic join and to specify algorithms which correctly process this information. Furthermore, an admission control might be necessary in case a joining receiver cannot obtain enough information to guarantee for the processing of synchronization. This remains to be subject of further research, too.

A *static* multicast group is defined as a multicast group that does not allow for the dynamic join. The synchronization as presented in this paper is based on the assumptions of static multicast groups. Static multicast groups can be simulated if in case of dynamic join the new receiver is brought in the same state as if it was present from the creation of the multicast group on.

2.5 Control Commands

User commands like *play, fast forward, stop, pause* are useful in some multimedia application scenarios like Video on Demand or Audio Players. A *control command* is defined as a user command influencing the course of the data presentation/processing in a distributed application. A control command may change the valid time interval of presentation units. This requires special treatment by the synchronization method. Many user commands result in control commands, like play, some, like resize a video screen, do not have an influence on the course of the presentation and thus are not relevant to synchronization. Due to control commands, the presentation schedule may no longer be a one-to-one mapping of the media schedule but the result of a function taking the media schedule as well as the control commands and the times of their calls as arguments.

Especially in case of inter-stream synchronization control commands are a very crucial issue. In case the commands have to be executed on the sender's side, the control command itself is transmitted from receiver to other receivers and sender. As the transmission of control commands itself suffers a delay that may be different for each flow, the synchronization method must pay special attention that the control command is performed at the same point of time in the media schedule for all flows that are concerned by this command. If this is not provided, the flows fall out of synchronization resulting in loss of synchronization. This synchronization error cannot be corrected unless treating each flow independently

from each other until the flows are re-synchronized. To do so, a central component has to control the re-synchronization of the data flows.

In case of point-to-multipoint and multipoint-to-multipoint communication a *floor-control* is necessary to determine which receiver is allowed to specify supported control commands that may influence the presentation sequence of the data stream(s). Within this document we assume a suitable floor-control in case of point-to-multipoint and multipoint-to-multipoint communication. This aspect is not further discussed here. If multiple receivers may trigger control commands, the performed sequence of control commands in all senders must be the same. If not, loss of synchronization may occur.

The *advancement of time*, either registration, sending or reference time always is linear during a presentation. This means, that the later displayed/presented/processed/sent presentation unit is always younger than the former one, even if the (logically) first presentation unit is displayed after a (logically) later presentation unit during the presentation, e.g. after calling the *rewind* command.

3 Related Work

Synchronization issues in multimedia systems as well as in distributed multimedia systems have been and are widely discussed in the literature. We do not claim to provide a complete overview of all published work dealing with synchronization in (distributed) multimedia systems. Instead, in this chapter we provide an overview of the related work that influenced the report on hand by in short explaining relevant issues.

As the published work treats different aspects of synchronization, it is structured here according to these aspects treated. For this reason some of the references are mentioned more than once.

The table in appendix B provides a brief overview of the related work.

3.1 Specification of Synchronization

Synchronization in multimedia systems refers to the maintenance and establishment of relations between different media units. These relations can be based on temporal or content information. The relations must be described and defined to be handled within a computer systems. A lot of work has been done to develop methods specifying synchronization relevant relations in multimedia systems. In general, the specifications are based on temporal relations between information units.

To do so, a notion of time is required. In 1972 Hamblin wrote an article about using instants and intervals to describe the time of events [Ham72].

Allen further developed the notion of intervals and defined 13 basic temporal relations between intervals [All91] [All83]: before, meets, overlaps, during, starts, and finishes with their inverses and equal.

[WR94] introduces another interval-based time representation. First, the basic temporal frameworks which are point-based or interval-based are introduced and evaluated. The authors then introduce their time representation which is interval based and contains the 10 basic interval relations: before, cobegin, coend, beforeendof, while, cross, delayed, startin, endin, and overlaps.

Petri Nets have been proved to be accurate to describe the synchronization relationships between different media. [Lit93] uses a *timed Petri Net (TPN)* to specify the synchronization relationships between different media units.

[LG90] defines the *object composition Petri Net (OCPN)* that is based on timed Petri Nets. It allows for the specification of all temporal intervals specified by [All83]. This Petri Net model has been used as basis for several specification schemes.

In [QWG93], e.g., the OCPN model is extended to the *eXtended object composition Petri Net (XOCPN)* allowing for specifying resource setup, resource release and inter-stream synchronization points within finer-grain objects.

The *real-time synchronization model (RTSM)* is also based on Petri Nets and an extension to the OCPN [YH96]. Its firing rules allow to *force* the advancement of time in case of one stream arriving too late. Thus, the RTSM is not only a model to describe the synchronization relationships but also the behavior in case of late coming media units/streams, i.e. in a specific error case.

To specify synchronization relationships between media, [ASC96] [ASCR95] propose the *interoperable Petri Nets (IPN)* that are based on colored *high level Petri Nets (HLPN)*. They allow for the specification of distributed applications as well as for the specification of accurate and fuzzy scenarios.

[dCCdSSC92] defines synchronization relevant terms based on the set theory. After introducing the basic terminology, intra- and inter-stream synchronization is defined. The authors rely upon a model of five different functional blocks in a multimedia system. The flow and bundle transfer in their model is always uni-cast.

[SKD95] [SKD96] aim to support synchronization of media segments of unknown and variable duration as well as to support synchronization in case of user interactions. Their synchronization approach supports only a very coarse grain synchronization between the begin and end points of media units. To do so, barrier and enabler conditions are defined. Intra-stream synchronization must be supported by another system.

[RH96] defines media clocks as well as a media clock hierarchy. The media clocks contain both, information on the local system clock as well as information on the media schedule. The defined media clocks allow for specifying the playback direction (forwards or backwards) as well as the playback speed.

[Ste90] introduces synchronization properties and basic terms in local multimedia systems. He relies on an object-based model of a multimedia system, the so-called *Multimedia Object System* comprising activities, multimedia objects and interactions. Basic synchronization characteristics are defined as well as enhancements for multimedia. The synchronization specification is based on reference points.

[PLL96] defines a *temporal reference framework* for the modeling of temporal information. For this purpose, the terms of a temporal scenario and of a model of time are introduced. A temporal scenario can be determined or in-determined. Five different models of time are relevant for the evaluated synchronization approaches: quantative dates, qualitative dates, qualitative instants, qualitative and quantative intervals. [PLL96] presents also an analysis and a comparison of different approaches to specify synchronization relationships.

[BS96], [ME93] and [SN95] describe a four layer model for synchronization. Each layer fulfills specified tasks and offers this task to its upper layer. Applications may be written to operate above each of these four layers. The specification layer serves for the creation of synchronization specifications. Layer three, the object layer, provides the synchronization between time-dependent and time-independent media. The stream layer synchronizes time-dependent media in what concerns intra- as well as inter-stream synchronization. The media layer provides a device-independent interface to different media.

[SN95] and [BS96] also present an overview on multimedia synchronization specification methods: interval-based, axes-based (global timer or virtual axes), control-flow based (basic hierarchical specification, reference points, timed Petri Nets), event-based and scripts.

3.2 Temporal Synchronization

Synchronization as the maintenance/establishment of temporal relationships is very intuitive. Therefore, a lot of synchronization approaches are based on temporal specifications of synchronization. The temporal relationships can be recorded simultaneously with the data, if they are not defined synthetically. Time-dependent data may be inter-stream synchronized using other approaches, but to correctly play out the data stream, i.e. to correctly perform intra-stream synchronization, time must be taken into account. Thus, at least a partly temporal synchronization scheme cannot be omitted.

[BHLM92] is based on reference points as synchronization specification method. They define an object model and distinguish two different basic objects. Dynamic basic objects are presented periodically and must be synchronized to this period whereas only the beginning and the end of static objects, e.g. text, is specified. These objects are only synchronized at the beginning and end point of their display.

[GBB96] describes a possibility to synchronize a data flow coming from a server array. This application is quite critical, as data for intra-stream synchronization already comes from different senders. A two-phase start-up protocol is defined. The first phase initializes the servers, the second phase realizes the synchronization. This synchronization method is based on the assumptions of no drift in clock speed, time stamps in packets and a bounded jitter.

[IT95] describes a synchronization method having the following requirements: no clock drift with regard to the speed and time stamps in the packets. The synchronization method allows for unbounded delays as well as for data that is not necessarily generated periodically. A master/slave concept is applied where the master stream only is intra-stream synchronized. Synchronization is described for tightly- and loosely-coupled streams.

[Lit93] describes a framework supporting time-dependent multimedia data. The synchronization specification is based on a TPN model and data is structured according to this model. Different synchronization relevant issues like: timing specifications, storage, play out scheduling and fine tuning of play out timing are addressed in this framework.

[LG90] introduces a data base architecture for multimedia data bases based on the OCPN model. The architecture includes the implementation of the synchronization scheme. An object retrieval and presentation algorithm is presented.

[RR93] presents synchronization based on feedback techniques. The synchronization is sender oriented, the resynchronization is done in the server. Multiple media phones may be supported by this synchronization scheme. Three different synchronization policies are described and evaluated: conservative, probabilistic and aggressive. Performance evaluations have shown that the best policy is the probabilistic one.

[RH95] describes the *Adaptive Synchronization Protocol (ASP)*. As the jitter in distributed applications is inevitable and must be smoothed out by buffering data, the goal of this protocol is to minimize the buffer requirements and, thus, the end-to-end delay. Streams are grouped together. There is one synchronization controller per synchronization group. The synchronization group follows the master/slave concept. The protocol is controlled by the sink which measures the experienced transfer delay and moves the play out point adaptively to the minimum delay producing a sufficiently low loss rate. ASP is subdivided into four sub-protocols: start-up protocol, buffer control protocol, master/slave synchronization protocol and master switching protocol. Two different synchronization policies are described: the minimum delay policy and the minimum loss policy. The protocol is based on the assumption of synchronized clocks.

[SSNA95] describes the *Concord* synchronization algorithm assuming a constant packet rate. Synchronization is achieved by controlling the play out buffer in the sink. In case of imperfect clocks (clock drift) the buffer is adjusted, which means that packets are doubled or thrown away regularly to avoid buffer under- and overflow.

[Sre94] presents a synchronization architecture based on agents. It follows the master/slave concept. Each stream is controlled by a stream agent which handles the intra-stream synchronization. A stream agent has an elastic buffer and is implemented within multiple threads. Two different methods for inter-stream synchronization are presented. The source-based model uses interleaving of data streams whereas a group agent coordinates the synchronization in the sink-based model. The advance of presentation time is either driven by real-time or by the stream time of a designated master stream. Stream agents trigger an event whenever the synchronization is violated or presentation relevant events occur. Applications may subscribe to the stream agents to get notifications on events and perform appropriate actions.

[YH96] presents the *Multimedia Synchronization Transport Protocol (MSTP)* that is based on a synchronization specification in RTSM. The MSTP manager is below the application and receives the QoS specifications as well as the RTSM for the data. One multimedia connection is established for every application. A sub-protocol runs for every medium. The MSTP manager in the sender sends the packets according to the RTSM specification. The MSTP manager in the receiver holds multiple variables to restore the synchronized order of the presentation units.

[ZLS96] defines an algorithm for inter-participant synchronization in real-time multimedia conferencing. A feedback technique is used, where the feedback serves as a means of reducing the playback time to a minimum. The time delay suffered from each packet may be different from one source to another. Also there may exist a phase offset between the different participants, even if they send with the same period. Therefore, packets belonging to the same set will most likely be received within a time window whose length depends on the phase offset of the sources, the frequency of the clocks as well as the network jitter. The synchronization is based on an estimation of the reference time and a calculation of the minimum waiting time for playback in the receiver as well as the minimum waiting time for sources with different periods. Using the presented synchronization method, the inter-participant synchronization can be kept if is is achieved at one time instance and if the exact frequency of all clocks of all sources is known to the receiver.

[MB95] uses the lower three layers of the four layer model for synchronization [BS96] and describes the possible implementation of synchronization in the MHEG standard.

3.3 Event-based Synchronization

Event-based synchronization is achieved by:

- triggering events whenever (re-)synchronization tasks need to be performed

- reacting on the events in real-time by performing the appropriate action

Temporal synchronization may also be regarded as event-based synchronization by defining the advancement in time as events triggered by some clock. The following three papers describe different aspects of the same event-based synchronization approach.

[PBCR95] introduces the event-based synchronization approach in an object oriented framework. Active and reactive objects are defined and their classes are described in detail. Multimedia objects, time-based activities as well as real-time support in the active object class are described. For reactive objects the creation as well as the interfaces between reactive objects and active objects are described. Example code for active and reactive objects is also presented. The method was implemented using the synchronous programming language ESTEREL.[5].

[SHH92] introduces a computational model for distributed applications and introduces the synchronous hypothesis following the specification of [Ber89]. Reactive objects are introduced as well as reactive systems and some basic concepts of ESTEREL.

[BCP+96] describes an object-oriented programming model based on reactive objects that implements the event-based synchronization approach using the synchronous programming language ESTEREL. The Chorus Micro Kernel was extended to provide the necessary system support for ESTEREL.

[Sre94] may also be regarded as event-based synchronization as streams trigger events whenever the synchronization is violated, i.e. whenever a resynchronization is necessary, or when presentation relevant events occur, i.e. when the data units have to be presented or removed.

3.4 Content-based Synchronization

A content-based synchronization approach for inter-stream synchronization is described in [OKR95]. The motivation lies in the significant amount of textual or imagery information being stored along with original multimedia data in content-based multimedia data bases to support efficient data retrieval. This information may be used to support synchronization, too. An object hierarchy is defined: composite object, segment object, event object, shot object and image set object. Content-based synchronization units have to be defined for an application as well as two different schedules. The operation schedule is used for presentation and specifies the overall sequential relations among the component objects. The synchronization schedule specifies the sequential and the simultaneous relationships between the component objects[6]. The paper presents some very interesting ideas but leaves some interesting questions open such as the topic of intra-stream synchronization.

3.5 Requirements

Perfect synchronization cannot be achieved as all real actions take time. For this reason, the goal of all synchronization schemes is always to provide a sufficiently good synchronization. There is a need to specify the characteristics of 'sufficiently good synchronization'.

Synchronization requirements with regard to jitter and skew are described in [Ste96]. Experiments were carried out to measure the human perception of jitter and skew. For lip synchronization an interval from $[-80\text{ ms}, +80\text{ ms}]$ for the skew can be regarded as 'in sync' region, whereas a skew lower than -160 ms or higher than +160 mswas almost always detected by the test persons. The experiments were carried out as well with a shoulder view as with a head or a body view. Similar experiments were also carried out to observe the 'in sync' region for pointer synchronization. [Ste96] provides a table of the accepted skews in different possible multimedia scenarios.

3.6 Classifications

Due to the many existing synchronization approaches, different classification schemes have been proposed. [BS96], [ME93] and [SN95] classify synchronization architectures according to the four layer model for

[5] more information on the ESTEREL language can be found in [BCG87] and [BG88]

[6] according to [SN95] this is a hierarchical specification

synchronization.

[SSF95] classifies inter-media synchronization schemes according to the following criteria:

- prerequisites: network wide clocks, availability of extra channels, alteration of data streams
- type of applications: stored data, real-time (live data), different sources
- costs: extra messages, additional channels, processing costs
- QoS requirements: delay/jitter, error, order of delivery

[PLL96] classifies synchronization approaches according to the underlying temporal scenario and temporal model, which determines the class of synchronization scenarios covered by the schemes.

In [dCCdSSC92] four different services for the intra-stream synchronization as well as two different services for the inter-stream synchronization are described. These services might be used as basis for a classification scheme.

3.7 System Model

Synchronization cannot be performed without looking at the environment of the application. This work takes a closer look to synchronization issues in distributed applications. Especially control commands as well as multi-party applications are considered. The presented synchronization method is theoretically described and theoretically evaluated. To do so, a model of the environment, a system model, is required. The system model is presented in chapter 4. The architecture of the model was mainly influenced by the project Da CaPo++ that was carried out at the Computer Engineering and Networks Laboratory (TIK) at the Swiss Federal Institute of Technology Zürich (ETHZ). An overview of this project is given in [SBC+97].

Within the project a basic synchronization mechanism was implemented as a diploma thesis [Bam97]. This work influenced the abstraction process of the Da CaPo++ architecture to the system model presented in chapter 4.

4 System Model

In this chapter we present the system model that serves as the basis for the synchronization method presented in chapter 5 and 6 and the analysis in chapter 7. The system model is flow based and abstracts from the concrete technical environment by logical units named components, describing separate parts of the environment. With this abstraction, we can formulate assumptions of our synchronization method which is based on characteristics of the components. Furthermore, this abstraction allows for calculating the synchronization quality in chapter 7.

Before taking a closer look on data flows and application requirements, the components of the system model are presented. We then formulate how relevant terms and characteristics described in chapter 2 are mapped within the system model. In chapter 8.1 a process model is described that may implement the presented system model and fulfills its real-time requirements.

As already stated, the system model was mainly influenced by the project Da CaPo++ [SBC+97].

4.1 Architecture

Five different *components* can be distinguished in the presented system model. Figure 4 shows these components for one single data flow.

The *network (NW) component* models the network link used within distributed applications. It comprises the physical link as well as the network protocol implemented on this link.

The *communication system (CS) component* models all communication support on the machine. This comprises the network access (use of the interface of the network protocol), and the handling of data buffers that are to be sent and of data packets that are to be received, as assembling, segmentation, encryption, compression, decompression, etc.

The *data handling (DH) component* handles the data received from the peer and the data that is to be sent to the peer. Data handling comprises the capturing from devices, the output to devices, the reading and writing from/to files, the preparation of data so that it can be handled by the communication system component. The data handling component also can hand over received data to the application and take data from the application that is to be sent to the communication peer. All data that is not received from or forwarded to the application is taken from or written to devices.

The *device (Dev) component* models input and output devices. Input and output devices in our sense are all interfaces that can retrieve/capture data or display/save data. This comprises cameras, microphones, video cards, audio cards, monitors, speakers, files, etc. The performed actions of the device component are directly initiated/controlled by the data handling component.

The *application (Appl) component* models distributed applications. These applications may be peer-to-peer or multi-peer applications. Applications that provide user quality have requirements that have to be fulfilled by the system. In our system model applications can specify requirements on a flow-based level, i.e. for each data flow requirements can be specified that are to be fulfilled by network, communication system, device handling and device components. Those requirements are specified in terms of *Quality of Service (QoS)*. The application can specify these requirements at the *flow interface (FI)* in our model. Thus, below the flow interface the requirements can be guaranteed. The application component can directly send or receive data to/from the data handling component.

The *control (Ctrl) component* models the interface between application and data handling component. Out-of-band communication is possible between the control components of sender and receiver. The control component can invoke functionality in the data handling component. Applications may steer the data flow, e.g. by providing interactive user commands as fast forward and zoom. These commands are sent to the control component within control packets and then forwarded to the data handling component. If the data handling component cannot provide the demanded functionality, e.g. commands like a fast forward cannot be performed by the receiver but must be performed by the sender, or does not understand the control packet, the packet is sent to the communication peer's control component.

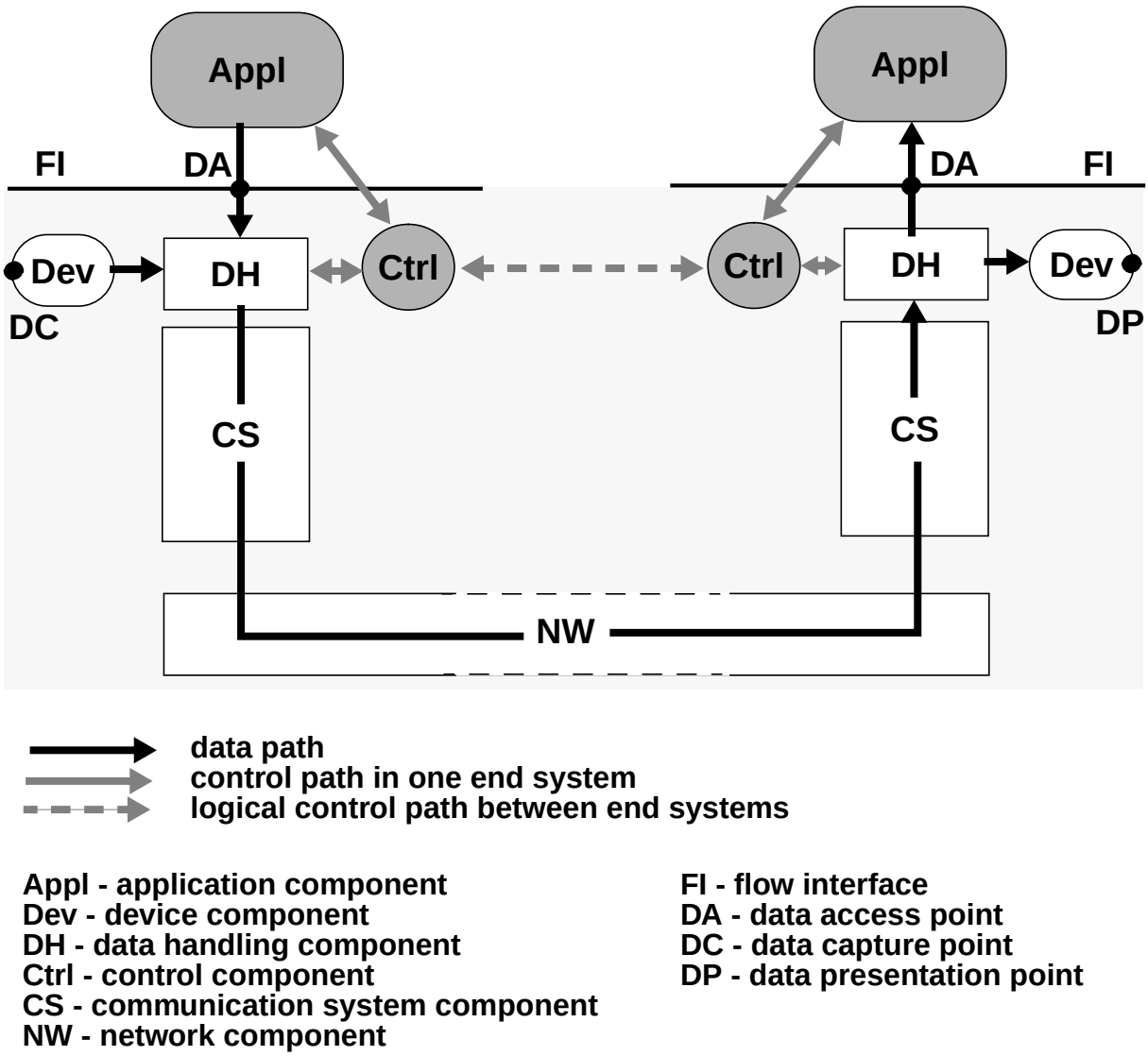

Figure 4: Model of One Data Flow in Distributed Applications

4.2 Data Flows

Our system model is flow-based. In figure 4 the components for one single data flow are shown. All flows are unidirectional, i.e. there is one sender and one or more receivers. In case of multiple receivers of the data flow, the data is multicasted. Flows are described by a media type (audio, text, video), a direction, and flow requirements (compare with [CS97]). As flow requirements can be specified, the data flows in the system model are connection oriented, i.e. requirements may be guaranteed.

Isochronous, i.e. time dependent media, as audio and video will in general be sent from a device component to another device component, e.g. from a camera to the monitor. Data that is sent by the application component at the data access point is originally time independent. If it is sent during a multimedia application, it has to be synchronized with the currently displayed/captured multimedia data. Thus, it becomes time dependent. As the time this data is sent is not known in advance, it has to be time stamped with the sending time, i.e. the time point at which the data handling component received this data to be sent is added as a time-stamp to the data. The sending time may also be translated to a reference time, if the data handling component of the time independent stream keeps track of the advancement in time during the presentation.

Multimedia applications in general consist of different data flows. For each flow a connection has to

be established and the specific flow requirements have to be specified at the flow interface.

Intra-stream synchronization only relies on information on one single data flow. Thus, all relevant actions can easily be performed within the components handling one data flow. Inter-stream synchronization requires information on multiple flows. Our system model, strictly separating flows and only allowing communication between the control components of flows via the application, aggravates the inter-stream synchronization.[7] In chapter 6 a synchronization method is proposed that guarantees for the inter-stream synchronization in case of well-known delay bounds for the communication channel. But inter-stream synchronization is critical in the given model if control commands are allowed. The problem is not completely solved within this document, even if section 6.3 proposes some possible solutions. All solutions as described in this report depend on additional characteristics of the system compared to the basic inter-stream synchronization method.

A similar, problematic situation arises automatically as soon as data flows are sent from different sources. Synchronization of different data streams may be performed by:

- the sources that have to synchronize the start and end of presentation, the advancement in time as well as the control commands. This adds additional communication links between the sources. Especially if sources support multiple applications at the same moment, as may do Video Servers e.g., the in this solution produced overhead might become critical.
- the receiver. This leads to solutions as described within this document.

The presented system model allows for easy intra-stream synchronization using the described components and some minimal control by the application. The inter-stream method as presented in this document works for the system model and can thus be applied in scenarios with different senders. But the inter-stream method cannot be realized in all environments as it is specific to the system model.

Both, inter- and intra-stream synchronization is designed using the time-stamping synchronization approach, i.e. no synchronization information is transmitted out-of-band, in spite of the fact that inter-stream synchronization has to be maintained in case of performing control commands. In this case information on the control commands must be transmitted as out-of-band data, too.

A similar stream model with independent streams is also used in [RH95] where the communication system, network and parts of the data handling component is comprised in the transmission path. In this model the network and the communication system component are modeled as transmission path.

4.3 Application Requirements

For every flow the application can specify *flow requirements* at the flow interfaces. The flow requirements are guaranteed below the flow interface, i.e. within the shaped area of figure 4 on page 18. Flow requirements, or *Quality of Service* can be of different types.

- *Flow characteristics:* The flow characteristics consist of data type and device type. The data type denotes the data that is to be transmitted, i.e. audio, video, text, graphics, e.g. and, thus, defines, among others, if the data itself is time-dependent or not. The device type specifies the device that is needed by the flow. If data is created or received directly by the application, the application is specified instead of a device (compare [CS97]).
- *Delay d:* The delay is defined as the time that elapses between data input on the sending side and data output on the receiving side. The data input point may be the *data access (DA) point* between application component and data handling component. Its input time is defined as the point of time the application writes data to be sent to the data handling unit. The data input point may also be the *data capture (DC) point* of the device component. In this case the input time is defined as the point of time the data is captured in the device component. The data output can be the data access point between data handling component and the application component or the *data presentation (DP) point* in the device. The output time is defined as the point of time the data

[7] The same characteristic allows, on the other hand, for the easy modeling of applications with multiple senders.

handling component gives the data to the application, or, respectively, the point of time the device presents the data, e.g. plays the audio or displays the video frame. Both, input as well as output time refer to the first bit of a presentation unit. We, thus, rely on a first-bit-first-bit delay.

The delay as defined within our system model comprises the network delay and the data processing time below the flow interface of sender and receiver. It also includes the packet delay if several PUs are stuffed into a single data packet. The delay that can be specified as a flow requirement is an end-to-end delay between DA or DC on the sender's side and DA or DP on the receiver's side.

The delay can be specified as an absolute value, e. g. 0.3 ms, which can denote a maximum, average, or exact delay, or it can be specified in form of a delay distribution. A delay distribution allows for specifying a probability $p(d)$ with which a specified delay is kept, i.e. $p(d)$ is the probability that the experienced delay is less or equal than d.

- *Jitter J:* The delay can be variable depending on current network and machine load. The jitter is the maximum difference of two delays of two successive packets. The jitter is specified as an absolute value and denotes the delay difference between packets that can be tolerated by the application. The jitter between two packets is not defined if one of those packets gets lost.

- *Throughput th:* The throughput is the number of bits per second that can be transmitted from any sender to a receiver, i.e. from the input point to the output point. This flow requirement denotes the throughput required by the application and, thus, refers to user data and does not include the control data. The throughput must be guaranteed by the network component by offering a physical link with a sufficiently high bitrate as well as by the end-systems that have to process the data in time.

- *Bit error probability $p(b_{err})$ and frame loss rate:* The bit error probability denotes the probability that one bit suffers a bit error between sender and receiver and serves as an indicator for the reliability of the network connection. There is a bit error rate specified for physical links that describes the rate of corrupted bits. As in a network, corrupted bits are not isolated whole data packets or cells (in ATM) are affected and get lost. If multiple data packets or cells are part of one data frame[8] and just one of the packets or cells gets lost, the frame gets lost, too. To decrease the call loss rate or packet loss rate visible by the application, protocols like *forward error correction* can be applied [Bie92]. On the application level a frame loss rate can be directly perceived. In case of time-dependent data the frame loss rate is increased by data that arrives too late in the receiver and thus cannot be presented in time or by data that is not processed in time in the end-system.

- *Frame rate/sample rate:* The frame rate denotes the number of video frames captured/presented per second and the sample rate denotes the number of audio samples captured/played per second. The frame number is specified as a number in the application requirements and denotes the number of frames that are shall be presented to the user per second. It does not refer to any characteristics of compression schemes.

Flow requirements are negotiated between sender and receiver. They are receiver oriented and, thus, the sender has to guarantee at least receiver demanded requirements. In multicast flows it may be possible that different receivers have different flow requirements specified, especially in what concerns delay, jitter and bit error probability. Flow requirements in receivers cannot be more stringent than those requirements supported by the sender. Connections between sender and receiver(s) can only be established if the flow requirements of all parties can be fulfilled. This must be guaranteed by an adequate admission control for each flow.

The system model of this paper assumes that these requirements can be guaranteed.

Especially we assume that end-systems can handle timer events in real time and that adequate admission control mechanisms guarantee that data flows can be processed according to their specifications. Paragraph 8.1 describes how this might technically be implemented.

[8]Data frame in this context refers to the data unit that is send in the data handling component. It is not to be confused with a video frame. a data frame may consist of multiple PUs.

4.4 Multi-party Applications

In the system model, one stream demands for one physical connection following the application requirements for this specific flow. The connection may be a unicast or a multicast connection. In this section the design of multi-party applications within the system model is shortly described.

A point-to-point application is an application that contains one or more single data flows between the two end-systems. The point-to-point application type is the easiest in what concerns intra-stream as well as inter-stream synchronization as both methods can be based on one single set of flow requirements for each flow.

A multipoint-to-point application is an application receiving multiple flows from at least 2 different senders. The flows are sent using unicast connections, i.e. using point-to-point connections between the receiver and each sender.

A point-to-multipoint application consists of one or more flows that are sent from one sender to multiple receivers using multicast. In case of multicasting flow requirements fulfilled by the sender delimit the possible flow requirements in the receivers. A multicast group may be a dynamic group, but the system model does not allow for negotiating or renegotiating the requirements leading to changes of the sender's flow parameters.

A multipoint-to-multipoint application is an application that consists of at least two flows coming from at least two different senders and being sent to the same multicast group. The flows are transmitted via multicast connections to the receivers.

4.5 Control Commands

In the presented system model, user commands are passed from the application to the control component. The control component passes the user command to the data handling component. There the command is evaluated. If the command can be performed locally and does not influence the course of the presentation, like decreasing audio, the data handling component processes this command. Otherwise the user command is a control command. Control commands have an influence on the data course and, thus, on the synchronization. They have to be passed to the sender's data handling or control component to be performed. Control commands may get parameters from the application to allow for maintaining inter-stream synchronization.

Control commands are transmitted via a so-called *out-of-band communication* link between receiver and sender. In case the sender has to send some additional information besides time-stamps to allow for maintaining inter-stream synchronization in presence of control commands, this information also is send via the out-of-band link.

4.6 Synchronization Errors

As set out in section 2.3 synchronization errors can have different reasons. For the synchronization analysis as described in the remaining part of this report, the following assumptions have been made:

- The technical environment implementing the system model is reliable. Errors, thus, cannot result from unreliable systems nor from errors in the modules implementing the synchronization.
- The clocks in the different end-systems are synchronized with respect to their speed. The maximum clock drift is bounded by δ_c.

Remaining synchronization errors may result from:

- Late arrival of PUs: PUs may still arrive too late to be presented within their valid time interval. The probability for this case is given directly with the violation probability in case of bounded delay. In case of a given delay distribution, this probability can be computed by $p(d) = 1 - p(x|x \leq d)$ where $p(d)$ is the probability that the delay is longer than d. If there is no given knowledge on the delay, the probability of the late arrival of PUs is not known.

- Loss of PUs: The probability of PU "losses" is indicated by the probability of erroneous PUs. This probability can be calculated with the bit error probability and the expected value of the length of a PU taken from the distribution of the PU length.[9]

Based on these assumptions a synchronization method for intra-stream and for inter-stream synchronization is presented and evaluated in the following chapters.

Each component in the system model fulfills a well defined functionality and can be looked at separately. This allows for the easy abstraction of the technical environment and the implementation of the functionality as well as for the description of the component by parameters describing its specific characteristics. The demands of different synchronization algorithms concerning the environment can be described in terms of values for these component parameters. This allows for classifying synchronization algorithms. As the parameters describing the components are mathematically treatable they can also be used to investigate quality issues for synchronization algorithms.

[9] If the length of PUs is not known, a value must be assumed that serves as an upper bound.

5 Intra-stream Synchronization

As already stated, the intra-stream synchronization has to be provided within the data stream, i.e. within a flow. In general, asynchrony may not only be added during the transmission but at any possible processing point. For this reason, the synchronization should be performed in the end system as near to the data displaying or data processing point as possible. In this context, "near" has both meanings, locally, no unnecessary inter-process communication nor data copy operations should be necessary to display the data, as well as temporally, synchronization should be done directly before presenting the data. The best point to do so in the system model is within the data handling component. The data handling component can communicate with the device component, and exchange control messages, and, thus, also error messages, with the application as well as with the peer data handling component (virtually via the control components in both sides).

Concerning the intra-stream synchronization of PUs in a data stream, three cases can be distinguished:

1. The presentation of the PU itself requires a specified time interval that can be obtained by the data specification. The device component controls that the PU is presented correctly within this time interval. The data is a continuous data stream in the sense, that after the PU is presented the next PU has to be presented immediately. In this case it is sufficient for the intra-stream synchronization to control the start time of the presentation of PUs. Examples for such data streams are audio streams and video streams.

2. The presentation of a PU requires 'zero' time in the sense that the time to present the PU is a time point and not an interval. The PU then remains presented until the next PU is presented. Thus, the user perceptible presentation time lasts from the time point the device presents the PU until the presentation of the next PU. For the purpose of intra-stream synchronization, it is sufficient to specify the presentation time for the PUs. Examples for such data streams are image sequences, were each image is displayed and remains on the display unit until the next image is displayed (to emulate a video data stream), or the sound of a clock, when every second a 'tic' sound PU is generated and presented.

3. The presentation of a PU is defined by a start and an end time. If the start time of PU $i+1$ is the end time of PU i, case one or case 2 is valid. If the end time of PU i is different from the start time of PU $i+1$, a media schedule is required to specify the start and end time of each PU in the data stream. We can define an *end PU* as a PU that contains no data to present but a time stamp. This PU does nothing but stopping the presentation of the currently presented PU by presenting zero data and, thus, clearing the data being presented before. In case of data that is live captured and transmitted, the end PU needs to be transmitted separately, in case of a given media schedule the end PU may be transmitted together with the data PU that shall be stopped indicating the end PU's representation time or a sending time that is calculated by the sender.

The problem of intra-stream synchronization can now be defined as the problem of synchronizing the start times of different PUs (including end PUs) of the same stream.

Definition 1 Let $t_{r,n}$ denote the reference time of PU n and $t_{p,n}$ denote the presentation[10] time of PU n. t_n denotes the time difference between the first and the nth PU in the media schedule. We assume no clock drift. A stream is synchronized with respect to intra-stream synchronization if:

$$\forall n : t_{r,n} = t_{r,0} + t_n \Rightarrow t_{p,n} = t_{p,0} + t_n \tag{1}$$

The asynchrony Δ in case of intra-stream synchronization is defined by:

$$\Delta = \max_n(|t_{p,n} - (t_{p,0} + t_n)|) \tag{2}$$

[10] The presentation time is the time point the data is available to the user/application. If the data handling and/or device component need some constant time t_x to prepare data for presentation, the presentation has to be triggered at $t_{p,i} - t_x$. We can substitute this by $t'_{p,i}$. Thus, all presented results remain valid with the newly defined presentation time. In case of a variant t_x, the variance adds as additional asynchrony to the asynchrony of synchronization.

This definition corresponds to Little's definition of intra-stream synchronization in [Lit93].

On the following pages the intra-stream synchronization is presented in case of point-to-point connections, as this is the easiest case. We then elaborate how intra-stream synchronization can be designed in case of multi-party distributed applications as well as in case of applications supporting control commands.

5.1 Intra-Stream Synchronization: Point-to-point

First assuming a constant delay, intra-stream synchronization is discussed. The assumption is weakened and afterwards a intra-stream synchronization based on a delay bound is described before examining a general case.

5.1.1 Constant Delay

First a constant delay is assumed. Intra-stream synchronization can be performed most easily when presentation units have to be presented periodically. Let ω denote this period. In this case there is no need for a media schedule. The basic intra-stream synchronization method for periodic data is performed as follows:

- Ordering aspect: Each PU to be transmitted gets an (increasing) PU number assigned to for the transmission. The receiver knows which PU to present next to maintain the order and guarantees that PUs are only presented when their PU number is higher than the number of all PUs presented before.

- Timing aspect: The timing aspect of synchronization can be implemented using timers. Whenever the timer sends one (periodical) time-out signal the next presentation unit has to be displayed. The term next refers to the next PU number and not to the next PU that was received. If this PU is not available, a synchronization error occurs (late arrival or loss of a PU). The application specifies its synchronization requirement in form of a maximal accepted asynchrony. Let this asynchrony be Δ. Let's assume that the first PU is captured at time $t_{s,0}$. The receiver displays this first PU at time $t_{p,0}$, where $t_{p,0} = t_{rc,0}$ and $t_{rc,0}$ is the time the PU was received. So the receiver can save the difference δ between the both times, which shall remain constant for all PUs to preserve the time order of the presentation of the PUs. That is: $\forall i : \delta = t_{p,i} - t_{s,i} = t_{p,0} - t_{s,0}$. Assuming clocks that do not have drifting speeds, PU number i has to be presented within the interval

$$[t_{p,0} + i/\omega - \Delta_b, t_{p,0} + i/\omega + \Delta - \Delta_b] \tag{3}$$

  to assure the maximum defined asynchrony of Δ. Δ_b denotes the asynchrony accepted before the optimal presentation/processing time. In general, Δ_b will be defined as $\Delta_b = \Delta/2$.

- Error cases: As mentioned in section 2.3 and 4.6, two possible error cases can occur within the system model and thus have to be treated in case of intra-stream synchronization. A PU may reach the receiver side after the information in the PU has to be displayed, i.e. the information can only be displayed late in time. This case occurs with the probability of $1 - p_a$, where p_a is the probability that a PU arrives in time. To retain the temporal relationship within the stream, those PUs have to be skipped and the following PUs have to be displayed in time. Mechanisms how to skip PUs and what may be done in such cases, like *restricted blocking*, etc. are presented in [Ste90], [SN95].

  The second error case occurs when a PU arrives corrupted or not at all. This case occurs with the probability of $p(b_{err}) \cdot l$, where $p(b_{err})$ denotes the bit error probability and l denotes the length of the PU.[11] In this case, the information sent within the PU cannot be displayed at all without retransmitting it. If the retransmission cannot be provided in time, the retransmitted PU has to be

[11] In distributed environments PUs are transmitted with communication protocol information (addresses, starting delimiter, etc) in form of PDUs. Of course, the protocol information itself may be erroneous, too. For this reason the bit error probability as used here, will be a function of the bit error probability of the communication subsystem and the probability that the PDU containing the PU(s) may not be delivered due to bit errors in protocol information.

also skipped. As already mentioned, the error probability can be decreased by methods of Forward Error Correction.

This is the basic intra-stream synchronization mechanism with its error cases. On the following pages we only discuss differences to this basic mechanism. The paragraph on error cases remains valid.

It is also possible, that data has to be synchronized that is not to be displayed periodically. We have to distinguish two cases. First, this data is captured live at the sender's side. A possible application is a picture phone, where audio data only is transmitted in case the user speaks. While capturing the data we can also record the time, the data was captured, and send it to the receiver, i.e. we use time-stamps indicating the sending time. Let's assume that the first PU is captured at time $t_{s,0}$. The receiver displays this first PU at time $t_{p,0}$. So the receiver can save the difference between the both times, which has to be a constant time difference, as receiver clock and sender clock are synchronized in clock speed. Let δ denote that difference. Then each PU captured at time $t_{s,i}$ has to be presented at the receivers side within

$$[t_{s,i} + \delta - \Delta_b, t_{s,i} + \delta + \Delta - \Delta_b] \tag{4}$$

to guarantee for the maximum accepted asynchrony. The capturing time $t_{s,i}$ must be transmitted from sender to receiver. In case the PU arrives late, the receiver has to skip it, as is in case of periodically presented data.

The second case of non periodic data is stored data. In this case, a media schedule providing the reference time-stamp for each PU has to be stored together with the data. The reference time denotes the relative time of the PU in the data stream where reference time 0 is associated to the first PU.

The media schedule may be sent to the receiver's site before the data transmission is started or at the begin of the data transmission. In this case, PU i must be presented at the receiver's side within

$$[t_{p,0} + t_{r,i} - \Delta_b, t_{p,0} + t_{r,i} + \Delta - \Delta_b] \tag{5}$$

where $t_{r,i}$ is the time indicated in the media schedule for PU i.

In case the media schedule is not transmitted to the receiver the PU i must be sent at time $t_{s,i} = t_{s,0} + t_{r,i}$. Here we have the same scenario as in case of live capturing non periodical PUs. This case also is given, if the media schedule is directly stored together with the data, i.e. each PU is stored together with its reference time.

To increase the probability of in time arrival of PUs p_a and thus decrease the error probability of late arrival of PUs $1 - p_a$ data that is requested in the future may be transmitted at the beginning of data transmission and stored in the receiver. The decrease of the error probability depends on the buffer size S_B in the receiver and the absolute delay distribution which limits the probability of erroneous transmitted PUs. There are two different methods that lead both to a peak in the transmission rate at the beginning of data transmission:

- All data required to fill the buffer on the receiver's site is sent before data transmission is started. This increases the initial delay of the data transmission. In presence of control commands like fast forward and fast rewind, the response time would be increased, too, as the buffer has to be re-filled before the presentation at the receiver can be started after a control command was triggered. If we take this response time into account, we have to increase the delay the system can guarantee which would result in a decrease of quality. For this reason we will not further elaborate on this method. On the other hand, the control commands might be performed locally. In this case, the problem of synchronization results into synchronization in a non-distributed application which is not the focus of this report.

- When the receiver receives the first PU it starts with presentation of the data. During a certain time of the beginning of the transmission[12] not only the requested PU but also PUs that have to be presented in the future are transmitted until the buffer in the receiver is filled. This method results in the same response time to control commands as without pre-transmitting data. The

[12] This time may be of fixed length or of variable length depending on network and/or system load.

required data bandwidth at the beginning of a presentation or after control commands is increased. In presence of control commands data may have been transmitted in vain. This method results in a decreased error probability of late arrival. In our further reflections we assume a well known error probability of late arrival of PUs, thus this method may be included in our scheme.

To guarantee intra-stream synchronization, the PUs have to be presented within the intervals as indicated in equation (3) to (5). The following discussion is based on these two intervals and the results in this section and discusses only additional issues for intra-stream synchronization.

5.1.2 Variable but Bounded Delay

In general we do not have a constant delay being guaranteed by the underlying network. For this reason we weaken our assumption and require a bounded delay only. A given network connection providing a bounded delay can be characterized by the following parameters: d_{min} and d_{max} denoting the minimum and maximum delay respectively, as well as $J = d_{max} - d_{min}$ denoting the jitter. To assure the intra-stream synchronization of data, the effect of jitter must be smoothed out by buffering PUs (compare [RH95]). To do so, the presentation of the first PU must be delayed so that is suffers the maximum delay.

The starting time for the presentation is calculated as follows:

$$t_{p,0} = t_{rc,0} + d_{max} - d_{min} = t_{rc,0} + J \tag{6}$$

where $t_{rc,0}$ denotes the time the first PU was received. Moreover we calculate δ as

$$\delta = t_{p,0} - t_{s,0} = (t_{rc,0} - t_{s,0}) + d_{max} - d_{min} = (t_{rc,0} - t_{s,0}) + J \tag{7}$$

As $t_{rc,i} = t_{p,i}\, \forall i$ in case of constant delay, we can state that the presentation of a PU within a data stream must be delayed for J time units to store enough PUs to smooth out the effects of jitter. The initial delay calculated in equation (6) is minimal in case that the first PU suffered minimal delay. In case the PU suffered maximal delay, the presentation could be started at once, as not additional delay has to be added to smooth out jitter. Because no global time is assumed, it cannot be determined if the PU has suffered a maximal delay and, thus, potentially unnecessary additional delay has to be added before starting the presentation to actually smooth out the jitter.

The required amount of buffered PUs to smooth out jitter is

$$B = \lceil 2 \cdot (d_{max} - d_{min})\omega \rceil = \lceil 2 \cdot J\, \omega \rceil \tag{8}$$

in case of periodic data and in case of non periodic data

$$B = \left\lceil \frac{2 \cdot (d_{max} - d_{min})}{\underline{\omega}} \right\rceil = \left\lceil \frac{2 \cdot J}{\underline{\omega}} \right\rceil \tag{9}$$

whereas $\underline{\omega}$ denotes the minimum time difference of the capturing/presentation time of two successive PUs, $\underline{\omega} = \min(t_{r,i}, t_{r,i+1}), \forall i$. In equation (9) B denotes a maximum bound for buffer required to guarantee the smoothing out of jitter. The factor 2 in equation 8 and 9 is necessary as the first presentation unit may suffer maximum delay and in addition be delayed for the jitter due to the starting time of the presentation (see equation (6)). The buffer size must be large enough to guarantee that no PU is lost due to buffer overflow in this case.

The buffer size can be computed by multiplying B with the number of bits in one PU. Given a constant PU size the buffer size of B multiplied with the PU size denotes the exact required buffer size to smooth out jitter. In case of a variable PU size the maximum number of bits in one PU is taken to compute the required buffer size in case of periodic data. The result then denotes an upper bound for the buffer size. In case of non periodic data both results denote an upper bound for buffer size as (9) already computes an upper bound.

Equation (8) and (9) may result in higher buffer sizes than required, but they guarantee that the calculated buffer size will be sufficient to smooth out the effects of jitter.

5.1.3 General Case

The system model provides communication paths guaranteeing for user specified QoS. Delay requirements may be specified as one of those QoS criteria. For this reason we can assume that the provided communication infrastructure can guarantee for a maximum delay. In this case intra-stream synchronization can be performed as described above. If it is not possible due to the underlying communication network, e.g. the Internet, to install a communication protocol guaranteeing for a bounded delay there are two possible consequences:

Delay distribution The communication system allows for assumping a statistical distribution of delays occurring for the PUs. This assumption might be reasonable for the Internet in a local environment. Depending on this distribution, the probability $p(d|(d \leq d_l)) = p(d_l)$ denoting the probability that the delay a packet suffers of is smaller than d_l can be calculated. $1 - p(d_l)$ denotes the probability that the experienced delay is longer than d_l, i.e. the probability that packets arrive too late to be presented within the specified synchronization time criteria, if a delay bound of d_l was assumed. This is defined to be a synchronization error. For this reason we can make the following statement:

In case of unbounded delay with a given delay distribution, we can choose a value d_b as to be a delay bound for the synchronization method in case of bounded delay, that guarantees for a maximum acceptable synchronization error probability, so that the synchronization error probability does not exceed the value specified as an application requirement.

No assumption on the delay If the system cannot even guarantee for a delay distribution allowing for the calculation of delay violation probabilities in case of an assumed delay bound, we cannot provide any guarantees concerning synchronization. Nevertheless, the intra-stream synchronization method as described above can be applied and may result in sufficient good synchronization quality in most cases, but no guarantees on the synchronization error probabilities can be given.

5.2 Point-to-multipoint, Multipoint-to-point, and Multipoint-to-multipoint

The above described method to synchronize one data flow in a point-to-point connection can be adapted to point-to-multipoint, multipoint-to-point as well as to multipoint-to-multipoint connections.

Intra-stream synchronization always refers to one single data flow which is described by QoS parameters, i.e. flow requirements, at the flow interface. Depending on those parameters, a connection is negotiated and established between the peers. All involved components must implement these requirements. Therefore, the intra-stream synchronization method as presented above can be applied by all receiving data handling components. The most crucial point are the control commands.

In what concerns the control commands in point-to-multipoint connections, a slight addition has to be done to the above described synchronization method. If a control command was issued by one receiver and changes the data flow, other receivers might generate synchronization errors, as they do have no knowledge on this control command. For this reason, control commands must be communicated to all receivers in the multicast group in case of a point-to-multipoint data stream.

The multipoint-to-point as well as the multipoint-to-multipoint connection scenario can only be implemented with at least two senders, which implies at least two different data flows. For this reason, these scenarios are not relevant for intra-stream synchronization. In what concerns intra-stream synchronization, point-to-point and point-to-multipoint connections cover all relevant cases in our system model.

The buffer requirements computed in equation (8) to (9) as well as the starting time for the presentation in equation (6) and δ in (7) are not only valid in point-to-point data streams, but also in point-to-multipoint data streams.

5.3 Control Commands

In case of stored multimedia data, applications may allow for control commands influencing the sequence of PUs to be transmitted and presented. The commands are performed on the sending side. Those commands may be (but are not restricted to) *forward, fast forward, rewind, fast rewind, play, stop,* and *pause* and may be designed in different ways examined here:

1. The period ω of the PUs remains the same but PUs are skipped (this may be the case in *fast rewind* and *fast forward* in an image sequence, e.g.): in this case the receiving data handling component needs not to perform any action. The sending side skips the PUs as specified within the command and sends the resulting sequence of PUs to the receiver following the given period ω for the data stream. The receiving side displays the incoming PUs with the specified period not taking into account that a control command is being performed.

2. PUs are skipped without a specified period ω: in this case the receiving data handling component has to control the time-stamps of the incoming packets to correctly initialize the timer. This has also to be done in case of non periodic data transmitted without any control commands being called by the user. So, in this case, too, the receiving side does not need to take into account the performing of the control command.

3. The presentation of PUs needs to be stopped or started: in this case the control command results in starting or stopping of the presentation of PUs. This is, of course, the case when performing the *start, stop, pause* commands but it also may be the case in performing the *fast forward, fast rewind* command for given media not allowing for simply skipping PUs and displaying the remaining ones like audio. In this case the receiving side needs to know of these commands not to generate a synchronization error in case of no more incoming PUs when the presentation needs to be stopped or in case of incoming PUs when the presentation is (re-)started. This can be easily provided as all control commands issued by the user or the application are passed by the receiving control component to be forwarded to the sending control component. In spite of the knowledge of the commands, the timer needs to be initialized correctly as in case 2.

In conclusion we can state, that the only necessary adaption of the synchronization mechanism to control commands consists in the forwarding of the control command from the calling receiver to all other receivers.

5.4 Summary

In this chapter a method was described to design intra-stream synchronization. The method is based on the introduced system model. Intra-stream synchronization as described in this chapter only requires clocks that are synchronized with respect to their speed.

In real world scenarios, clocks will not advance exactly the same way. But protocols like NTP can guarantee for a maximum clock drift [Mil91] and, thus, also for a maximum drift in the advancement of clocks. This drift adds to the asynchrony as defined in equation (2).

As evaluated in chapter 7 the synchronization method discussed in the report guarantees for a specified asynchrony. To do so, the method is based on guarantees as defined within the system model. Such guarantees are the reaction to timer events in real-time, the presentation of multimedia data in real time as well as a specified end-to-end delay bound. If these guarantees cannot be provided by the system, the synchronization method may nevertheless be applied as described above. But it will only result in a synchronization quality as good as possible and not in a guaranteed quality.

6 Inter-stream Synchronization

The intra-stream synchronization examined in the last chapter is concerned with the synchronized presentation of PUs belonging to the same data flow. PUs belonging to different data flows may also require to be synchronized, this is done by inter-stream synchronization. Before discussing how inter-stream synchronization can be performed within the system model, some basic terms have to be defined.

A *synchronization group* is defined as the set of all data flows that have to be synchronized together. For the synchronization method as presented in this report we assume the synchronization group to be *static*. A synchronization group is defined to be static if from the beginning to the end of the transmission and presentation of the data all flows belonging to this group are well known. It is, thus, not possible to dynamically add or delete flows to/from a static synchronization group. A *non static* synchronization group requires some kind of admission control determining if a given data flow can be synchronized and thus be added to a synchronization group. Furthermore, the current state of the synchronization group has to be determined to correctly initialize and add the new flow. How this might be done in a general approach remains subject of further investigations and will not be further elaborated in this report.

Often the concept of master streams is used ([IT95]). A *master stream* is the reference data stream, other streams are synchronized to. If the PU of the master stream is late, no PUs are presented, even if all PUs of the *slave streams* are available. If the PU of the master stream is available in time, all available PUs are presented, even if some PUs of slave streams are lacking. The master stream concept used in [RH95] is dynamic, as streams can dynamically become masters or slaves (master switching sub-protocol) if they risk to violate application requirements.

The master stream approach requires communication between the different streams of one synchronization group or a central entity, like the synchronization controller in [RH95]. In the presented system model, communication between the display and buffer administrating units of the different data streams results in communication between data handling, device and control component of the different data flows. This communication is necessary at least from the master stream to the slave streams to determine, if a set of synchronous PUs has to be presented or not. Our system model does not provide any communication paths between different components not belonging to the same flow. Therefore, the master stream approach can only be implemented, if the communication from the master stream to the slave streams is provided by the application. This implies the communication from the master stream control component to the application as well as communication from the application to all slave stream control components. In our system model, real-time bounds are only guaranteed below the flow interface, i.e. in the device, data handling, communication system, network and control component, but not in the application. Realizing the master stream concept in the system model, a synchronized presentation of PUs can no longer be guaranteed in real-time but may add additional delay. For this reason, the master stream inter-stream synchronization approach is not further evaluated here.

Consequently, all data flows belonging to a synchronization group are treated in the same way within the synchronization approach that is discussed in this report. Before introducing the basis of the inter-stream synchronization method, inter-stream synchronization is defined.

Definition 2 Let i, j be two flows belonging to the same synchronization group S. Let $t_{i,r,n}$ denote the reference (r) time of PU n in the media schedule of flow i and $t_{i,p,n}$ denote the presentation time (p) of PU n of flow i. Let n, m denote numbers of PUs. There is no clock drift. The flows belonging to a synchronization group S are synchronized with respect to inter-stream synchronization, if

$$\forall i, j \in S, \forall n, m : t_{i,r,n} = t_{j,r,m} \Rightarrow t_{i,p,n} = t_{j,p,m} \tag{10}$$

The asynchrony Δ is defined as

$$\Delta = \max_{n} \left(|t_{i,p,n} - t_{j,p,m}| \right) \quad \text{with} \quad t_{i,r,n} = t_{j,r,m} \tag{11}$$

Theorem 1 shows that the inter-stream synchronization can be achieved by synchronizing any two PUs of two flows that have to be presented concurrently, if the two flows are synchronized with respect to intra-stream synchronization and there is no clock drift.

Theorem 1 Let *i, j* be two flows of the same synchronization group S. We assume no clock drift. Both flows are synchronized with respect to intra-stream synchronization. Let n be the number of any PU in flow i and m be the number of any PU in flow j that have to be presented concurrently, i.e. $t_{i,r,n} = t_{j,r,m}$. The flows i and j are synchronized with respect to inter-stream synchronization, if

$$t_{i,p,n} = t_{j,p,m} \tag{12}$$

Proof With equation 1 follows: $t_{i,r,n} = t_{i,r,0} + t_{i,n} = t_{j,r,0} + t_{j,m} = t_{j,r,m}$. Thus, $t_{i,r,0} - t_{j,r,0} = t_{j,m} - t_{i,n}$. Let *m'* and *n'* be any two PUs with $t_{i,r,n'} = t_{j,r,m'}$. With equation 1 follows $t_{i,r,0} - t_{j,r,0} = t_{j,m} - t_{i,n} = t_{j,m'} - t_{i,n'}$. As the flows i and j are synchronized with respect to intra-stream synchronization, follows: $t_{i,p,n'} = t_{i,p,0} + t_{i,n'} = t_{i,p,n} + t_{i,n'} - t_{i,n}$ and the same for $t_{j,p,m'}$. Thus, the difference in the presentation time of PU *n'* and PU *m'* is:

$$\begin{aligned} t_{i,p,n'} - t_{j,p,m'} &= t_{i,p,n} + t_{i,n'} - t_{i,n} - (t_{j,p,m} + t_{j,m'} - t_{j,m}) \\ &= t_{i,p,n} - t_{j,p,m} + t_{j,m} - t_{i,n} - (t_{j,m'} - t_{i,n'}) \\ &= t_{i,p,n} - t_{j,p,m} = 0 \end{aligned}$$

That is, the flows i and j are synchronized with respect to inter-stream synchronization. q.e.d.

Theorem 1 is valid especially for the first PUs that have to be displayed concurrently.

The temporal intervals of Allen as described in [All83] are a general approach to describe temporal relationships between intervals. PUs in the context of synchronization always are presented during a time interval. Many methods have been proposed to describe synchronization requirements and, thus, the time relationships between the presentation intervals of PUs. This article supposes a given media schedule that specifies when and for which time each PU is to be displayed[13]. To show that above definition of inter-stream synchronization is sufficient to describe the inter-stream synchronization requirement in every possible media schedule, we define the construct of a *synchronization point*. A synchronization point is a time point that may be either within or without a presentation time interval. It does not cause any action but just specifies a synchronization time point. It is defined in the media schedule. A synchronization point gets a time assigned to exactly like PUs. It is $t_{i,r,m}$ the time specified in the media schedule either for a PU m or a synchronization point m and $t_{i,p,m}$ the presentation time of PUm or the time point the synchronization point m is valid in the presentation. As there is no clock drift and as the streams are synchronized with respect to intra-stream synchronization, it is in flow i: $\forall n : t_{i,r,n} = t_{i,r,0} + t_{i,n} \Rightarrow t_{i,p,n} = t_{i,p,0} + t_{i,n}$, where n is the number of a PU or a synchronization point.

Theorem 1 is valid not only for all PUs but also for the set of all PUs and synchronization points. The proof is analog.

We can now formulate the transitivity of the inter-stream synchronization relationship.

Theorem 2 Let *i, j, k* be any two flows of a synchronization group. If i and j are synchronized with respect to inter-stream synchronization and j and k are synchronized with respect to inter-stream synchronization, then i and k are synchronized with respect to inter-stream synchronization.

Proof We add a set of synchronization points to the media schedule of each flow, so that if one flow has a PU or an end PU with media schedule time t_x, the other two flows have a PU, end PU, or synchronization point with the same media schedule time t_x. As the flows *i and j*, or *j and k* are inter-stream synchronized, we have $\forall n, m : t_{i,r,n} = t_{j,r,m} \Rightarrow t_{i,p,n} = t_{j,p,m}$ and $\forall n, m : t_{j,r,n} = t_{k,r,m} \Rightarrow t_{j,p,n} = t_{k,p,m}$ with *n, m* any PUs or synchronization points in the flows. It follows: $\forall n, m, l : t_{i,r,n} = t_{j,r,m} = t_{k,r,l} \Rightarrow t_{i,p,n} = t_{j,p,m} = t_{k,p,l}$. Thus, the flows i and k are synchronized. q.e.d.

[13] In case of periodic data the media schedule is implicitly given. In case of live captured data the media schedule is captured together with the data.

Theorem 3 Given a synchronization group S. We have no clock drift. The flows of the synchronization group are synchronized with respect to inter-stream synchronization, if they are synchronized with respect to intra-stream synchronization and if for PUs m, n of any two flows i, j we have: n, m minimal with $t_{i,r,m} = t_{j,r,n} \Rightarrow t_{i,p,m} = t_{j,p,n}$.

Proof follows directly from Theorem 1 and 2.

The inter-stream synchronization method presented in this article is based in Theorem 3, i.e. we assume that the flows of a synchronization group are synchronized with respect to intra-stream synchronization and then synchronize the first PUs of any two flows of the synchronization group that have to be presented together.

The synchronization method as described in [IT95] is based on assumptions as formulated in Theorem 1 to 3 as well as on clocks that have the same advancement in time, i.e. no clock drift.

6.1 Inter-stream Synchronization: Multipoint-to-point

The in this report discussed synchronization approach is stream based and just assumes no clock drift. For this reason the point-to-point and the multipoint-to-multipoint case can be treated in the same way even if multipoint-to-point scenarios seem to be a tough case.

6.1.1 Variable but Bounded Delay

First we examine the case that the first PUs of all data flows belonging to the synchronization group have to be presented concurrently. Given be a static synchronization group with n different data streams belonging to this group. Let presentation unit $PU_{i,0}$ be the first PU of data stream i. For each data stream the maximum delay $d_{i,max}$ is known. The following method guarantees the inter-stream synchronization:

1. The application sends a start message to the data handling component of each flow of the synchronization group with a local start time that is $t_s = 2 \cdot \max_i(d_{i,max}) + \Delta_{t,Appl} + \Delta_{t,i}$, whereas $\Delta_{t,Appl}$ denotes a maximum limit for the necessary processing time in the application.[14] $\Delta_{t,i}$ is the delay caused by the time the peer data handling components need to recognize the command and to start the requested functionality, which is assumed to be bounded and the bound is assumed to be known.

2. The data handling component belonging to flow i sends the start command to the peer data handling component via the control component at time $t_{i,s} = t_s - 2 \cdot d_{i,max} + \Delta_{t,i}$, where $d_{i,max}$ is known from the flow specification (the QoS requirements for the corresponding flow) or is the delay that can be guaranteed by the communication link with a given probability as described on page 27.

3. The sender start sending their data as soon as they receive the start command.

4. The receiver presents the first PU of flow i at time $t_{i,p,0} \in [t_s - \Delta_b, t_s + \Delta - \Delta_b]$ with $0 \leq \Delta_b \leq \Delta$, where Δ denotes the maximum accepted asynchrony for the inter-stream synchronization. As the sender has sent this PU at time $t \leq t_s - d_{i,max}$ the PU is available in time at the receiver's site.

The in time presentation of all successive PUs is assured by the intra-stream synchronization (see Theorem 1), i.e. above described method achieves inter-stream synchronization.

We now assume data flows where not all first PUs have to be processed concurrently. Those data flows have to be specified by a media schedule enabling to determine the relations for inter-stream synchronization. Thus, for each PU the reference time is specified. Above method can be applied to assure inter-stream synchronization with the following change: the start message initializes a reference

[14] This includes the communication of the start message to the data handling component as well as the processing time in the local data handling component.

zero time in every sending data handling component. The PUs then are sent with respect to their media schedule and the reference zero time. This task is fulfilled by the intra-stream synchronization.[15]

If the intra-stream synchronization is guaranteed for all data flows, the above described method guarantees the inter-stream synchronization of the data flows on account of Theorem 3. Each receiver device component starts the data presentation at time t_s.[16] Therefore, the presentation of $PU_{i,0}$ starts at the same moment for all i.[17] As all data flows are synchronized regarding to the intra-stream synchronization, the different data flows will remain synchronized during time.

The buffer requirement of each data flow is a bit higher than in equation 8 and 9. Applying above described inter-stream synchronization method, the maximum number of PUs in the buffer must also include the jitter in transmitting the start command as well as $\Delta_{t,i}$[18]. Therefore the maximum number of PUs of flow i in the buffer is calculated as

$$B_i = \lceil (2\,(d_{i,max} - d_{i,min}) + \Delta_{t,i})\,\omega \rceil = \lceil (2\,J_i + \Delta_{t,i})\,\omega \rceil \tag{13}$$

and in case of non periodic data:

$$B_i = \left\lceil \frac{2\,(d_{i,max} - d_{i,min}) + \Delta_{t,i}}{\underline{\omega}} \right\rceil = \left\lceil \frac{2\,J_i + \Delta_{t,i}}{\underline{\omega}} \right\rceil \tag{14}$$

The synchronization method takes into account that different streams may suffer different delays and that different sources may need to start at different times. Moreover, we do not assume a well-known delay of the first PU neither a well-known delay for the start command. Under these circumstances, the required buffer is minimal to smooth out the different possible jitters and guarantee the inter-stream synchronization.

6.1.2 General Case

In case of a non-bounded delay, the presented inter-stream synchronization method can only be performed if one of the following assumptions is valid:

- A delay distribution is given
- Globally synchronized clocks guarantee for a global time in all peers

Delay distribution In case of a given delay distribution the synchronization method for a bounded delay can be applied choosing a suitable delay d_l with a delay violation probability of $p(d_l)$ so that the maximum specified synchronization error probability can be guaranteed. This case is similar to the delay distribution described in section 5.1.3.

Globally synchronized clocks In case of globally synchronized clocks in the receiver all PUs from all flows having the same time stamp can be displayed together. To do so and to assure intra-stream synchronization the receiver has to assume a delay δ for the transmission such that a PU with time stamp t_i is displayed at time $t_i + \delta$. In case of no given delay distribution, this method is not suitable for a QoS guaranteeing application as no guarantees for a maximum synchronization error probability can be provided.

If there is neither a specified delay distribution nor are there globally synchronized clocks, above described synchronization method can also be applied. To do so the reference time must be sent together with every PU so that the receiver knows when the PUs have to be displayed. This results in the method

[15] This method is also valid in case of live captured data.

[16] With a maximum guaranteed asynchrony of Δ.

[17] With a maximum guaranteed asynchrony of Δ .

[18] This term is taken into account for the calculation of buffer requirements as the time for recognizing and performing control commands can vary. If this variation is not exactly known, what we assume, the use of $\Delta_{t,i}$ in the formula calculates an upper bound for the buffer requirement.

of global clocks but is only valid in case of pre-recorded data. It is not suitable to synchronize two live data streams coming from two different senders (e.g. audio and video from a speaking person are recorded in different specialized computers with different local times and sent to the receiver). In case that non of the two assumptions above is valid, inter-stream synchrony, however, can no longer be guaranteed by a synchronization method as the synchronization error cannot be bounded.

6.2 Point-to-point, Point-to-multipoint, and Multipoint-to-multipoint

The point-to-point communication is a special case of the multipoint-to-point communication. Therefore, the above described synchronization method can also be applied to the point-to-point communication scenario. Due to its characteristics, the point-to-point communication also offers another possibility to maintain inter-stream synchronization in case of control commands. All sending data handling components are located in the same end system. For this reason it is possible, that the sending application on the end system implements a synchronization feature to guarantee for a synchronized performance of control commands (especially the start command). Whenever a control command is received by a data handling component on the sending side, it is passed to the application without executing it at once. The application, then, sends this command to all data handling components specifying a start time for the control command. The data flows remain to be sent synchronously and, thus, do not fall out of synchronization. This is a special method only valid in case of one sender and is thus not further examined here.

In case of point-to-multipoint communications the inter-stream synchronization may be performed on the sending side as described above. The performed control command has to be sent to all receiving data handling components to avoid false error messages (see section 5.2).

In case of multipoint-to-multipoint communication the synchronization method as described in section 6.1 can be applied if the sending data handling components forward the control command that has to be performed to all receiving data handling components. This avoids false synchronization error messages in the receivers as described in section 5.2.

6.3 Control Commands

Whenever a control command is performed for different streams, these streams may fall out of synchronization. Such commands include, but are not restricted to, e.g. *play, pause, stop, fast forward, fast rewind, forward, rewind.* The inter-stream synchronization in case of applications using control commands is presented in this paragraph.

Assuming a bounded delay as in section 6.1 the inter-stream synchronization requires media schedule time-stamps. The control command itself is also transmitted with a certain not-exactly known delay to the sender's data handling component which then starts the performance of the command. As the delay is not exactly known, it cannot be assured that all senders start the control command at the same time $t_{m,i}$ within the media schedule. This results in asynchrony. This asynchrony is bounded by $f(d_j - J_j, d_k + J_k)$ where j is the connection with the minimum delay and k the connection with the maximum delay. The function $f(x, y)$[19] depends on the control command and the implementation of the command and calculates the asynchrony. The asynchrony resulting from control commands may accumulate with every call of a control command. For this reason, control commands are a very critical issue for inter-stream synchronization. The following methods can maintain the inter-stream synchronization in case of control commands:

- To avoid the cumulation of asynchrony, the application may specify when the control command exactly has to be performed. This time must be specified as reference time (media schedule time) and be sent together with the control command to the sender(s). This requires, that the application itself knows at any point of time, a control command may be specified, which media schedule time stamp the current PUs have, resulting in a lot of communication between the data handling components of the flows and the application. For this reason, this approach results in some overhead. However,

[19] The definition of the function $f(x, y)$ remains subject of further investigations.

the method is quite robust in what concerns the delay of the transmission of control commands. Even if the command arrives too late at the sender, i.e. after the specified execution time, the sender may compute the current state in the media schedule if the command was performed in time and thus send the correct data after the command was received. This results in a resynchronization with the other data streams.

- To maintain inter-stream synchronization in case of control commands, the following protocol could be applied. The application specifies the control command that has to be performed. The command is sent to all senders. The senders respond with the earliest possible media time to execute the command. This time takes into account twice the delay between sender and receiver (or an assumption on the delay) as well as the processing time in the receiving application. The application collects the answers and defines the maximum of the media times as the start time of the command. If a sender receives the command with its specified start time after this time has elapsed, it has to re-synchronize the affected data stream by determining the actual media time if the command has been performed in time and sending the corresponding PU(s).

- Assuming a bounded delay for the data transmission as well as a well-known constant delay for control commands that are transmitted via the control components, the transmission of the media schedule time stamp with the PUs is no longer necessary. The application can specify when each sender has to receive the control command. Let t denote this time. The receiving data handling component sends the control command via the control component at t minus the constant delay for the communication with the peer's control-component. This guarantees that all senders start performing the control command at the same time. In consequence the data streams remain synchronized at the receiver's side.

- Another method for the inter-stream synchronization assumes globally synchronized clocks that guarantee for the same time in the senders as well as the receiver. In this scenario it is possible, that the application specifies a time when the control command has to be performed and that all receiving data handling components send this time together with the control command to the senders. As all communication peers have the same time, the control command will be performed simultaneously in all senders.[20]Thus, the intra-stream synchronization of the data flows guarantees that the inter-stream synchronization can be maintained. Assuming a bounded delay which is known to the application, the application can specify the starting time for each control command so that all sending data handling components receive this command in time.

Inter-stream synchronization in case of control commands can be maintained in different ways, above described methods add additional requirements to the system model or additional costs.

6.4 Summary

In this chapter a method was described to design and implement inter-stream synchronization. The method is based on the system model. Inter-stream synchronization as described in this chapter only requires clocks that are synchronized with respect to their speed. Similar restrictions hold as for the intra-stream synchronization. As for the intra-stream synchronization the inter-stream synchronization is based on the assumption that system guarantees are provided.

The described inter-stream synchronization is based on the assumption of data streams that are synchronized with respect to intra-stream synchronization.

The maintenance of control commands in the case of inter-stream synchronization demands some more guarantees to be fulfills depending on the possibility chosen from the catalogue presented in section 6.3.

[20]In case the control command arrives too late in one sender, the sender can re-synchronize with the other senders due to the specified start time of the command.

7 Quality of Synchronization

In today's applications the quality provided to the user is a critical point for user acceptance of the application. A lot of work is done how the user quality may be specified in terms of *Quality of Service (QoS)* parameters and how those may be provided in distributed systems (e.g. [Sti96]). Our system model supports QoS applications by offering to applications the possibility to specify their requirements for each data flow in terms of Quality of Service parameters/application requirements (see section 4.3). These application requirements can be guaranteed below the flow interface. Users also may be interested in specifying synchronization quality.

At the European Networking Center (ENC) at Heidelberg experiments have been carried out to measure quality requirements for synchronization with respect to *asynchrony*. These experiments were restricted to measure the *skew* (inter-stream asynchrony) between different data streams (namely video and audio as well as telepointer and audio) and the *jitter* (intra-stream asynchrony) within one data stream, to determine thresholds that are not perceived as being disturbing by the user. These results are published in [Ste96], which states that the user perceives an asynchrony of more than 80ms (video ahead of audio) or more than 40ms (audio ahead of video) as disturbing. Only jitter and skew are used to define the synchronization quality.

Our understanding of synchronization quality is more extensive. Not only the asynchrony must be taken into account but as well the *probability of synchronization errors*. The synchronization error probability serves as a measurement of correctly synchronized PUs that are displayed within a specified time unit. In general, this number will be smaller than the rate of correctly transmitted PUs as late coming PUs are thrown away.

Another important criterion for QoS in synchronization is the *delay*. If the time a user has to wait for the execution of a command or for the start of a presentation exceeds a specified time span, it is perceived as a decrease of quality. However, synchronization methods may need to provide additional delay to achieve synchronization.

Providing synchronization with reasonable asynchrony, low synchronization error probability as well as low delay results in costs. One indicator for costs may be the *buffer requirements* of the synchronization scheme. Other criteria to measure cost of synchronization are to be investigated.

In this chapter the synchronization issues as presented in chapter 5 and 6 are evaluated with respect to the above mentioned criteria.

7.1 Maximum Asynchrony

In this section we will show, that a maximum asynchrony of Δ can be guaranteed if PUs are displayed in the interval of equation (3) to (5) respectively and if the PUs are inter-stream synchronized as described in chapter 6. To do so, we will calculate the asynchrony guaranteed by the intra-stream synchronization and the one guaranteed by the inter-stream synchronization. These asynchronies have to be added.

Definition 3 *The maximum asynchrony Δ_{ia} in case of intra-stream synchronization* is defined as follows: Let $t_{r,i}$ and $t_{r,i+1}$ denote the media schedule time (representation time), $t_{p,i}$ and $t_{p,i+1}$ denote the presentation time for PU i and *i+1*. Δ_{ia} is the maximum asynchrony in case of intra-stream synchronization if

$$\forall i : \max(0, t_{r,i+1} - t_{r,i} - \Delta_{ia}) \leq t_{p,i+1} - t_{p,i} \leq t_{r,i+1} - t_{r,i} + \Delta_{ia} \tag{15}$$

With the term $\max(0, t_{r,i+1} - t_{r,i} - \Delta_{ia})$ we avoid that a large term Δ_{ia} allows for violating the ordering aspect of intra-stream synchronization.

Theorem 4 The maximum asynchrony Δ_{ia} is guaranteed by the in section 5 proposed intra-stream synchronization method.

Proof With the proposed intra-stream synchronization method PU i is presented within the interval:
$[t_{p,0} + i/\omega - \Delta_b, t_{p,0} + i/\omega + \Delta_{ia} - \Delta_b]$ with $0 \leq \Delta_b \leq \Delta_{ia}$ in case of periodic data, and
$[t_{s,i} + \delta - \Delta_b, t_{s,i} + \delta + \Delta_{ia} - \Delta_b]$ with $0 \leq \Delta_b \leq \Delta_{ia}$ or
$[t_{p,0} + t_{r,i} - \Delta_b, t_{p,0} + t_{r,i} + \Delta_{ia} - \Delta_b]$ with $0 \leq \Delta_b \leq \Delta_{ia}$ respectively in case of non periodic data.
Part 1: $t_{p,i+1} - t_{p,i} \leq t_{r,i+1} - t_{r,i} + \Delta_{ia}$
By subtracting the lower bound of the presentation time of PU i, $t_{p,i}$, from the upper bound of the presentation time of PU *i+1*, $t_{p,i+1}$, we obtain: $1/\omega + \Delta_{ia}$, $t_{s,i+1} - t_{s,i} + \Delta_{ia}$ and $t_{r,i+1} - t_{r,i} + \Delta_{ia}$ as the maximum time difference of the presentation of two successive PUs. In the proposed synchronization method, the sending time $t_{s,i}$ is obtained according to the media schedule, thus, $t_{s,i+1} - t_{s,i} = t_{r,i+1} - t_{r,i}$. The period ω corresponds to the period in the media schedule, thus, $1/\omega = t_{r,i+1} - t_{r,i}\, \forall i$ in case of periodic data.
Therefore, it is $t_{p,i+1} - t_{p,i} \leq t_{r,i+1} - t_{r,i} + \Delta_{ia}\, \forall i$ and for all variants of the intra-stream synchronization method.
Part 2: $t_{r,i+1} - t_{r,i} - \Delta_{ia} \leq t_{p,i+1} - t_{p,i}$
By subtracting the upper bound of the presentation time of PU i, $t_{p,i}$, from the lower bound of the presentation time of PU *i+1*, $t_{p,i+1}$, we obtain: $1/\omega - \Delta_{ia}$, $t_{s,i+1} - t_{s,i} - \Delta_{ia}$ and $t_{r,i+1} - t_{r,i} - \Delta_{ia}$ as the minimum time difference of the presentation of two successive PUs. Due to $t_{s,i+1} - t_{s,i} = t_{r,i+1} - t_{r,i}$ and $1/\omega = t_{r,i+1} - t_{r,i}$ it is $t_{p,i+1} - t_{p,i} \geq t_{r,i+1} - t_{r,i} - \Delta_{ia}\, \forall i$ and for all variants of the intra-stream synchronization.
Part 3: $\max(0, t_{r,i+1} - t_{r,i} - \Delta_{ia}) \leq t_{p,i+1} - t_{p,i} \leq t_{r,i+1} - t_{r,i} + \Delta_{ia}$
If $0 \leq t_{r,i+1} - t_{r,i} - \Delta_{ia}$ the proof follows directly from part 1 and part 2.
If $t_{r,i+1} - t_{r,i} - \Delta_{ia} \leq q$ this results in a maximum defined asynchrony allowing for violating the ordering principle of synchronization. In other words, the accepted asynchrony may be larger than the difference of the reference time of two successive PUs. The synchronization method as described on page 24 guarantees the playout of PUs in the correct order. For this reason it is guaranteed that $t_{p,i+1} - t_{p,i} \geq 0$.

The synchronization method, thus, guarantees for a maximum intra-stream asynchrony of Δ_{ia}. q.e.d.

Definition 4 The *maximum asynchrony* Δ_{ie} *in case of inter-stream synchronization* is defined as follows: l and k are any two flows belonging to the same synchronization group. Let $t_{k,r,i}$ and $t_{l,r,j}$ be the time for PU i in stream l and PU j in stream k as specified in the media schedule (representation time), $t_{k,p,i}$ and $t_{l,p,j}$ be the presentation time for PU i in stream k and PU j in stream l. Then Δ_{ie} is the maximum asynchrony in case of inter-stream synchronization if

$$\forall k, l : t_{k,r,i} = t_{l,r,j} \Rightarrow |t_{k,p,i} - t_{l,p,j}| \leq \Delta_{ie} \tag{16}$$

Theorem 5 The in section 6 presented synchronization method guarantees for a maximum inter-stream asynchrony of Δ_{ie}.

Proof To evaluate the inter-stream asynchrony only, we assume an intra-stream asynchrony of 0. The inter-stream synchronization method guarantees that the first PUs (PUs or synchronization points) of any two flows of the synchronization method are displayed within the time interval $[t_s - \Delta_b, t_s + \Delta_{ie} - \Delta_b]$. Assuming that the PU of one flow is presented at the earliest time and the PU of the other flow is presented at the latest time, we obtain an asynchrony of those two PUs of Δ_{ie}. As the both streams are perfectly intra-stream synchronized, Theorem 1 limits the asynchrony of any two PUs of those two streams to Δ_{ie}. With Theorem 2 we can follow that the asynchrony of any two flows of the synchronization group is limited to Δ_{ie}. Thus, the maximum asynchrony is given by Δ_{ie}. q.e.d.

As discussed in section 6.3 this theorem is not always valid in case of control commands. For this reason the presented evaluation holds only under the assumption that no control commands are performed or that control commands are performed in a way that guarantees for uninterrupted inter-stream synchronization (e.g. by using global clocks).

Theorem 6 The proposed synchronization method guarantees for a maximum asynchrony of in total $\Delta = \Delta_{ia} + \Delta_{ie}$.

Proof Any two PUs of any two flows of the synchronization group are displayed with an intra-stream asynchrony of Δ_{ia} and an inter-stream asynchrony of Δ_{ie}. PU m of flow i is, thus, displayed in the time interval $[t_{i,p,0}+t_{i,r,m}-\Delta_{b,ia}-\Delta_{b,ie}, t_{i,p,0}+t_{i,r,m}+\Delta_{ia}-\Delta_{b,ia}+\Delta_{ie}-\Delta_{b,ie}]$ Thus, the maximum asynchrony of any two PUs n and m of any two flows i and j of the synchronization group with $t_{i,r,m} = t_{j,r,n}$ is calculated by subtracting the earliest possible presentation time of one PU from the latest possible presentation time of the other PU. This results in $\Delta = \Delta_{ia} + \Delta_{ie}$. q.e.d.

With the proposed synchronization method we can, thus, guarantee for a maximum asynchrony. The accepted asynchrony in intra-stream and inter-stream synchronization has to be chosen depending on the total maximum asynchrony and the costs for the synchronization method. Concerning these issues [Ste96] serves as a valuable indication to determine the accepted asynchrony.

7.1.1 Clock Drift

In the former discussion of synchronization issues, clocks without clock drift, with the same advancement in time, have been assumed. We now investigate what happens, if this assumption does not hold, which is the case in the real world.

To make clock drift tractable, we must assign a *value* to the clock drift. This is not so easy since we need a global reference time t_g to specify this value. For this reason we specify a global reference time with its advancement serving as a base to measure the clock drift.

First, we examine the case that clocks are regularly resynchronized. The accuracy of these clocks can be maintained to within a few milliseconds [Mil91] (this can be achieved by using specified protocols, e.g. NTP). The clock drift δ_c can be limited by this asynchrony.[21] δ_c takes the maximum value of the asynchrony of all clocks, whereas the asynchrony is measured with reference to the global reference time t_g. Calculating the maximum asynchrony for intra-stream synchronization in analogy to the proof of Theorem 4, the clock drift δ_c adds to the asynchrony. The same holds for the maximum asynchrony in case of inter-stream synchronization.

For this reason the maximum asynchrony in case of clock drift being *bounded* to δ_c is:

$$\Delta = \Delta_{ia} + \Delta_{ie} + 2 \cdot \delta_c \tag{17}$$

If the clock drift cannot be bounded, this may result in a constantly changing (maybe even increasing) time difference between the different clocks. For this reason, the clock drift is a function of time $\delta_c(t) = f_c(t)$. Also the asynchrony is a function of time. Based on equation (17) it follows:

$$\Delta(t) = \Delta_{ia} + \Delta_{ie} + 2 \cdot \delta_c(t) = \Delta_{ia} + \Delta_{ie} + 2 \cdot f_c(t) \tag{18}$$

Thus, the asynchrony becomes a function of time which is constantly changing (maybe increasing). This does not allow to guarantee for a maximum asynchrony in case of non bounded clock drift.

7.2 Synchronization Error Probability

As discussed in section 4.6 two possible cases result in synchronization errors:

- The PU is corrupted, i.e. bit errors occurred. This happens with the probability of: $p(b_{err}) \cdot l_{PU}$, where l_{PU} is the expected value for the length of one PU. $p(b_{err})$ denotes the bit error probability of user data which is derived of the bit error probability of the network and the data packet structure in the network. In case of a corrupted packet the data cannot be presented or does not at all arrive and the PU has to be skipped.

[21] This may not seem very straightforward as it assumes that the total asynchrony is always present. But this is a guaranteed upper bound. For this reason this bound can be used to calculate a maximum asynchrony.

- The PU arrives late which occurs with a probability of $p(d_l)$. In this case the PU cannot be presented in time and has to be skipped.

In case of constant or bounded delay being guaranteed by the communication subsystem, the two cases mentioned above appear the same to the application as those details are transparent to it.

As both synchronization errors may occur independently from each other, we can calculate the total synchronization error probability as:

$$p_{err} = p(b_{err}) \cdot l_{PU} + p(d_l) - p(b_{err}) \cdot l_{PU} \cdot p(d_l) \tag{19}$$

In case of a non-bounded delay, but a given delay distribution, equation (19) can be used to determine the "delay bound" $p(d_l)$ if a maximum synchronization error probability p_{err} was specified by the application:

$$p(d_l) \leq \frac{p_{err} - p(b_{err}) \cdot l_{PU}}{1 - p(b_{err}) \cdot l_{PU}} \tag{20}$$

That is, the upper delay bound is limited to $p(d_l)$ as calculated in equation (20) to guarantee a synchronization error probability of p_{err} maximum.

The probability $p(b_{err}) \cdot l_{PU}$ is given by the underlying network and, thus, cannot be influenced by the synchronization scheme. The probability $p(d_l)$ is dependent on the delay that can be specified at the flow interface. This probability, too, cannot be influenced by the synchronization scheme.

7.3 Delays

The user perceives two kinds of delays during data transfer:

- The *initial delay*: After a user started an application some time passes by before the data of the application really is received on the user's side. This time is called initial delay.
- The *round trip delay*: The round trip delay denotes the time that passes by after the user has triggered a user command and before the user perceives the effects of this user command.

Assuming that all control commands are handled in the same way by the application and the components as the start-up of the application described in section 6.1.1, the initial delay and the round trip delay are identical. They are given by:

$$d_{init} = 2 \cdot \max_i(d_{i,max}) + \Delta_{t,Appl} + \Delta_{t,i} \tag{21}$$

7.4 Buffer Requirements in the Receiver

Equation (8) and (9) calculated buffer requirements for intra-stream synchronization. For inter-stream synchronization the buffer size requirements are calculated in equation (13) and (14). The buffer size requirement for stream i for intra-stream synchronization as well as for inter-stream synchronization within the synchronization group is, thus, given for periodic data by:

$$B_i = \lceil (4\,J_i + \Delta_{t,i})\,\omega \rceil \tag{22}$$

and for non periodic data by:

$$B_i = \left\lceil \frac{4\,J_i + \Delta_{t,i}}{\omega} \right\rceil \tag{23}$$

J_i denotes the jitter in data transmission of flow i. If the jitter is zero, the buffer requirement is zero, too, as the buffer only is needed to smooth out the effects of jitter. In case of an unknown jitter but a variable delay bounded to a maximum delay or a delay distribution with a delay bound calculated by equation (20), the maximum delay can be taken instead of the jitter to calculate an upper bound for the buffer size requirement.

This buffer requirement exists for every stream being synchronized with the intra-stream and inter-stream synchronization method. Thus the total buffer requirement for a synchronization group S can be bounded as:

$$B_s \leq \sum_{i \in S} B_i \tag{24}$$

where i determines the streams in S.

B_s only denotes the buffer requirement needed to smooth out the effects of the different jitters. If data is sent faster than it is to be presented in the receiver, additional buffer is required to avoid buffer overflow.

7.5 Synchronization Quality

Synchronization quality depends at least on the quality criteria described above. Those criteria are not independent, they influence each other. E.g. the specified maximal initial delay influences the delay that may be specified below the flow interface. This delay determines, in case all other parameters are unchanged, the synchronization error probability (probability of late arrival of PUs) as well as the required maximum buffer size (in case no jitter is specified).

One crucial issue that is not elaborated here are the *costs* to provide synchronization, i.e. the communication and system costs to fulfill the flow requirements, as well as the costs for the buffer to smooth out the effects of jitter. There exist also interdependencies between Quality of Service below the flow interface and communication and system costs. All those factors need to be taken into account to specify an adequate, user oriented synchronization quality.

Above introduced criteria have to be examined if they present a valid notion for quality of synchronization and the interdependencies between the criteria need further investigation.

8 Implementation Architecture of Synchronization within the System Model

8.1 Process Model

All discussed synchronization relevant issues are based on the presented system model. For showing their validity, the system model must be implemented in a real-world environment. One possible way is presented here.

The presented device, data handling, control and communication system component are implemented within one process for each one flow. The network component models the access to the communication link and must be implemented as an independent process, too. To allow for the processes to handle their tasks in real-time, a *real-time scheduler* is required. Especially when a timer alarm is triggered, this scheduler guarantees that the timer handling task can be performed within a sufficient small time interval. The real-time scheduler allows for specificating time bounds and guarantees for all tasks being performed in time within the components. For this reason, guarantees on the end-to-end delay can be given in case of guaranteed network delays.

To allow for guarantees in end-systems, especially for real-time guarantees, an *admission control* must be provided. The admission control guarantees that time-critical processes or processes requiring non-shared resources, like a camera, are only allowed (admitted) if the required resources are available with the demanded guarantees.

8.2 Intra-stream Synchronization

The implementation of the intra-stream synchronization is based on the assumption that the sender sends the PUs in a synchronized manner so that the required synchronization information can be retrieved by the receiver out of the reference time stamps in the PUs. Furthermore, it is assumed, that the sender does not send the PUs faster than specified by the period or the media schedule, as this could lead to a buffer overflow in the receiver and would require further control information exchange between the data handling components.

As the presented synchronization method is time based and requires the presentation of PUs in specified time intervals, timers need to be used. These timers have to work accurately and the call-back routine for a timer must be processed in real-time. For this reason, a real-time scheduler, an operating system supporting components to operate in real-time mode as well as accurate timers are essential for the implementation of the synchronization method presented above (cf. section 5). As the sender has to send the PUs in a synchronized manner, real-time scheduling and timer management also has to be provided in the sender.

The following changes/extensions need to be implemented to enable the intra-stream synchronization for data streams, i.e. for data flows:

1. Time stamps are generated in the sender and are transmitted together with the PUs, where reference time 0 denotes the start time of the entire application.

2. The sender starts timers to send the PUs (a periodic timer in case of periodic PUs, a timer with a time-out time derived from the media schedule in case of non-periodic PUs) or sends PUs instantly, whenever they have been captured by the device component or received from the application.

3. The receiver needs to start timers to present PUs in time. Early arrived PUs are saved intermediately in the buffer, late PUs are dropped.

4. In case of periodic data, when the period is known to the receiving data handling component, the data handling component generates error messages to the application, whenever an expected PU is not being received in time. In case of non periodic data, error messages are generated and sent to the application, whenever a PU is received late. These error messages may serve as indicator to the user for the reliability of the given guarantees.

The intra-stream synchronization method as described here can easily be implemented. To enable intra-stream synchronization in presence of control commands for point-to-multipoint and multipoint-to-multipoint applications, the communication of control commands between the control components of sender and receivers has to be provided. Specifically all receivers have to be notified if one receiver calls a control command. The flow control between receivers, i.e. the administration which receiver may call which command at which time, is the task of the application and not of the synchronization process.

8.3 Inter-stream Synchronization

The inter-stream synchronization method described in this article can be integrated easily within an environment that implements the system model presented in section 4. We assume that the intra-stream synchronization is implemented (see section 8.2). Furthermore, we assume that the sender sends the reference time with each PU.

The application must implement inter-stream synchronization control, which, in an object-oriented environment, may be designed as a separate component (class) that implements this functionality. The application must have information about all flows that have to be synchronized as well as information on the delay bounds for each data flow. A start command is sent to all data handling components specifying the starting time t_s of the application, i.e. reference time 0. According to equation 21 t_s is at least $t_c + 2 \cdot d_{max} + \Delta_{t,Appl} + \Delta_{t,i}$, where t_c is the current system time, d_{max} is the maximum delay of all flows, $\Delta_{t,Appl}$ is an upper bound for the application processing time and $\Delta_{t,i}$ denotes an upper bound for the processing time in the data handling component. The data handling components forward the start command at time $t_s - 2 \cdot d_f + \Delta_{t,i}$, where d_f is the delay of the flow (see section 6.1.1).

In case of unreliable senders it is also possible to start a two phase start-up protocol. At time t_c a start message with time t_s is sent to the senders. The senders receive this message at the latest at time $t_c + \Delta_{t,Appl} + d_f$. In case one sender is not able to start sending the flow(s) in time, an error message is sent from the sending data handling component to the receiver(s). The error message is received by the receiving application(s) at the latest at time $t_c + 2 \cdot \Delta_{t,Appl} + 2 \cdot d_f$. If at least one error message was received, the application in every receiver sends stop commands to the data handling components of all flows. These commands must arrive in the sending data handling components before data has been sent. Thus, the two phase start-up protocol requires the start time to be specified at least as $t_s = t_c + 4 \cdot d_{max} + 3 \cdot \Delta_{t,Appl} + 3 \cdot \Delta_{t,i}$.

To maintain synchronization in case of control commands, the forth possibility as described in section 6.3 may be applied.

9 Summary and Future Work

In this report synchronization issues arising in distributed applications are discussed.

In the first part of the report issues relevant to synchronization, basic terms as well as basic synchronization mechanisms are defined and presented. Based on the terminology defined in this report two specific problem fields of synchronization in distributed applications are introduced: multi-party applications and control commands. Related work in different fields of synchronization is presented and shortly compared.

The second part of the report is based on a temporal view of synchronization. These synchronization issues are studied in detail and a notion of synchronization quality is presented.

Synchronization cannot be discussed without taking into account the environment and application requirements (e.g. , multicast) as these determine the tasks to be fulfilled by the synchronization method. One of these tasks is to smooth out the effect of jitter and is only required if jitter exists. For this reason a system model is introduced. The environment of our study is described by means of the system model. The system model is based on the assumption that service guarantees can be provided by the underlying network and by the end-system.

Intra-stream as well as inter-stream synchronization is defined formally in this report. This allows for the exact formulation of valid time intervals and, thus, for the analysis of synchronization quality. A synchronization method for intra-stream synchronization is presented. This method can be applied in a real-time environment guaranteeing a bounded delay/well-known end-to-end delay distribution as well as clocks without clock drift. This method is also applicable in case of control commands like, e.g., stop and fast forward. Furthermore, a synchronization method for inter-stream synchronization is presented. This method is based on delay bounds/well-known end-to-end delay distributions and speed synchronized clocks. Different possible solutions are presented on how inter-stream synchronization can be retained in presence of control commands.

We also investigate quantitatively the quality of the proposed synchronization scheme by calculating the maximum asynchrony as well as the synchronization error probability, buffer requirements, the round trip delay, and the initial delay for the synchronization scheme. We prove that the proposed synchronization method can guarantee an upper bound on asynchrony. The investigated parameters are part of our notion of synchronization quality.

The implementation architecture of the system model as well as of the synchronization method is described briefly. The description serves as an indication that synchronization as described in this document may be implemented in a real-world environment.

The presented work is a first step to a detailed investigation of synchronization issues in distributed multi-party applications that may support control commands and a step to a notion of quality of synchronization. Further work may be carried out in the following five topics:

- *Network/end-systems*: The abstract system model introduced in this report describes the network and end-system. Existing networks as well as end-systems have to be investigated and described by relevant parameters with the aim to complete the abstract system model to allow for a more detailed study of general synchronization schemes. Especially, standardized parameters should be used to describe requirements and prerequisites of a synchronization method.

- *Existing synchronization schemes*: The definitions provided in this report as well as the synchronization quality criteria defined, can be used to investigate existing synchronization schemes. This investigation would allow for a classification and evaluation of synchronization algorithms' achievements. Part of these classification criteria are parameters describing the network and the end-system.

- *Synchronization quality*: This report defines synchronization quality criteria. These are subject to further evaluations which also include the close study of interdependencies between the different criteria.

- *Multi-party applications*: The presented study of issues related to synchronization in multi-party applications has to be extended to multiparty dynamics. Especially, multicast groups allowing for

the dynamic join of receivers, and dynamic synchronization groups, allowing for senders and flows to join dynamically, have to be studied in further detail. This also includes the investigation of existing synchronization schemes with respect to their applicability to multi-party scenarios.

- *Control commands*: In this report possible solutions to synchronization in presence of control commands are presented, but this problem requires additional extensive studies. These studies shall comprise an exact definition of the asynchrony introduced by control commands as well as the investigation of necessary conditions for synchronization in presence of control commands.

Therefore, future work will have the following goals:

- To extend the concept of Quality of Service to synchronization and relevant parameters;
- To define a generic evaluation scheme for synchronization mechanisms;
- To define a framework for synchronization in distributed multi-party applications; and
- To implement/simulate an exemplary list of existing schemes or newly designed ones.

References

[All83] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[All91] J. F. Allen. Time and Time Again: The Many Ways to Represent Time. *International Journal of Intelligent Systems*, 6:341–355, 1991.

[ASC96] Y. Y. Al-Salqan and C. K. Chang. Temporal Relations and Synchronization Agents. *IEEE Multimedia*, 3(2):30–39, summer 1996.

[ASCR95] Y. Y. Al-Salqan, C. K. Chang, and Y. V. Reddy. MediaWare: On Multimedia Synchronization. In *Proc. of the Int. Conference on Multimedia Computing and Systems*, pages 150–156, Washington, D.C., May 15-18 1995.

[Bam97] M. Bamert. Synchronisation multimedialer Datenströme. Master's thesis, TIK, Swiss Federal Institute of Technology Zürich (ETHZ), March 1997.

[Bau97] D. Bauer. *A Multipoint Communication Architecture for End-to-End Quality of Service Guarantees*. PhD thesis, Swiss Federal Institute of Technology Zürich (ETHZ), April 1997.

[BCG87] G. Berry, Ph. Couronne, and G. Gonthier. Synchronous Programming of Reactive Systems: An Introduction to ESTEREL. Technical Report 647, INRIA, Sophia-Antipolis, France, May 1987.

[BCP+96] G. S. Blair, G. Coulson, M. Papathomas, Ph. Robin, J.-B. Stefani, F. Horn, and L. Hazard. A Programming Model and System Infrastructure for Real-Time Synchronization in Distributed Multimedia Systems. *IEEE Journal on Selected Areas in Communication*, 14(1):249–263, January 1996.

[Ber89] G. Berry. Real time programming: special purpose or general purpose languages. In G.X. Ritter, editor, *Information Processing 89*. IFIP, Elsevier Science Publishers B.V. (North-Holland), 1989.

[BG88] G. Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. Technical Report 842, INRIA, Sophia-Antipolis, France, May 1988.

[BHLM92] G. Blakowski, J. Hübel, U. Landgrehr, and M. Mühlhäuser. Tool support for the synchronization and presentation of distributed multimedia. *Computer Communications*, 15(10):611–618, December 1992.

[Bie92] E. Biersack. A Simulation Study of Forward Error Correction in ATM Networks. *ACM SIGCOMM Computer Communication Review*, 22(1):36–47, January 1992.

[BS96] G. Blakowski and R. Steinmetz. A Media Synchronization Survey: Reference Model, Specification, and Case Studies. *IEEE Journal on Selected Areas in Communications*, 14(1):5–35, January 1996.

[CDK94] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems*. Addison-Wesley, 1994.

[CS97] C. Conrad and B. Stiller. A QoS-based Application Programming Interface for Communication Middleware. In *SPIE for the Voice, Video, and Data Communication Symposium*, volume 3233, Dallas, Texas, USA, November 1997.

[dCCdSSC92] L. F. Rust da Costa Carmo, P. de Saqui-Sannes, and J.-P. Courtiat. Basic Synchronization Concepts in Multimedia Systems. In P. V. Rangan, editor, *Network and Operating System Support for Digital Audio and Video, 3rd International Workshop*, pages 94–105, La Jolla, California, USA, November 1992.

[GBB96] W. Geyer, C. Bernhardt, and E. Biersack. A Synchronization Scheme for Stored Multimedia Streams. In B. Butscher, E. Moeller, and H. Pusch, editors, *Interactive Distributed Multimedia Systems and Services, European Workshop IDMS'96*, pages 277–296, Berlin, Germany, March 1996. Springer.

[Ham72] C. L. Hamblin. Instants and Intervals. In J. F. Fraser, F. C. Haber, and G. H. Müller, editors, *The Study of Time: Proceedings of the First Conference of the International Society for teh Study of Time*, pages 324–331, Oberwolfach, West Germany, 1972. Springer Verlag.

[IT95] Y. Ishibashi and S. Tasaka. A Synchronization Mechanism for Continuous Media in Multimedia Communications. In *IEEE INFOCOM 95, The Conference on Computer Communications*, volume 3, pages 1010–1019, Boston, MA, April 1995.

[Lam78] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[LG90] T. D. C. Little and A. Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Areas in Communication*, 8(3):413 – 427, April 1990.

[Lit93] T. D. C. Little. A framework for synchronous delivery of time-dependent multimedia data. *Multimedia Systems*, (1):87 – 94, 1993.

[MB95] Th. Meyer-Boudnik. *Interaktive Multimedia-Präsentationen in offenen Systemen*, volume 365 of *Fortschritt-Berichte VDI. Reihe 10, Informatik/Kommunikationstechnik*. VDI-Verlag, 1995.

[ME93] T. Meyer and W. Effelsberg. Multimedia-Synchronisation: Eine vergleichende Analyse der bestehenden Methoden. In W. Effelsberg and K. Rothermel, editors, *Verteilte Multimedia Systeme*, pages 189–205, Stuttgart, Germany, 18./19. Februar 1993. GI/TG Arbeitstreffen.

[Mil91] D. Mills. Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[OKR95] D.-Y. Oh, S. S. Kumar, and P. V. Rangan. Content-based inter-media synchronization. In *Multimedia Computing and Networking 1995*, volume 2417, pages 202 – 214, San Jose, California, February 1995. SPIE - The International Society for Optical Engineering.

[PBCR95] M. Papathomas, G. S. Blair, G. Coulson, and P. Robin. Addressing the real-time synchronization requirements of multimedia in an object-oriented framework. In *Multimedia Computing and Networking 1995*, volume 2417, pages 190 – 201, San Jose, California, February 1995. SPIE - The International Society for Optical Engineering.

[PLL96] M. J. Pérez-Luque and T. D. C. Little. A Temporal Reference Framework for Multimedia Synchronization. *IEEE Journal on Selected Areas in Communications*, 14(1):36–51, January 1996.

[QWG93] N. U. Qazi, M. Woo, and A. Ghafoor. A Synchronization and Communication Model for Distributed Multimedia Objects. In *Proc. of the First ACM Conference on Multimedia*, pages 147–155. ACM Press, New York, August 1993.

[RH95] K. Rothermel and T. Helbig. An Adaptive Stream Synchronization Protocol. In *5th Int. Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, New Hampshire, USA, April 18-21 1995.

[RH96] K. Rothermel and T. Helbig. Clock Hierarchies: An Abstraction for Grouping and Controlling Media Streams. *IEEE Journal on Selected Areas in Communications*, 14(1):174–184, January 1996.

[RR93] S. Ramanathan and P. V. Rangan. Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks. *IEEE/ACM Transactions on Networking*, 1(2):246–259, April 1993.

[SBC+97] B. Stiller, D. Bauer, G. Caronni, C. Class, C. Conrad, B. Plattner, M. Vogt, and M. Waldvogel. Communication Support for Distributed Applications. Technical Report 25, TIK Institute, ETH Zurich, Switzerland, January 1997.

[SHH92] J.-B. Stefani, L. Hazard, and F. Hørn. Computational model for distributed multimedia applications based on a synchronous programming language. *Computer Communications*, 15(2), March 1992.

[SKD95] J. Schnepf, J. A. Konstan, and D. Du. Doing FLIPS: FLexible Interactive Presentation Synchronization. In *Proc. of the Int. Conference on Multimedia Computing and Systems*, pages 213–222, Washington, D.C., May 12-15 1995.

[SKD96] J. Schnepf, J. A. Konstan, and D. Du. Doing FLIPS: FLexible Interactive Presentation Synchronization. *IEEE Journal on Selected Areas in Communication*, 14(1):114–125, January 1996.

[SN95] R. Steinmetz and K. Nahrstedt. *Multimedia: Computing, Communications and Applications*, chapter 15. Synchronization, pages 567 – 670. Prentice Hall, P T R, 1995.

[Sre94] C. J. Sreenan. A Service Oriented Approach to Continuous Media Synchronization. In *IEEE INFOCOM '94*, volume 2, pages 936–943. IEEE, 1994.

[SSF95] V. Saparamadu, A. Senevirante, and M. Fry. A review of inter media synchronization schemes. In *Proceedings of the First Int. Conf. on Multi-Media Modeling*, pages 229–239, Singapore, November 1995.

[SSNA95] N. Shivakumar, C. J. Sreenan, B. Narendran, and P. Agrawal. The Concord Algorithm for Synchronization of Networked Multimedia Streams. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 31–40. IEEE Computer Society Press, May 15-18 1995.

[Ste90] R. Steinmetz. Synchronization Properties in Multimedia Systems. *IEEE Journal on Selected Areas in Communications*, 8(3):401 – 412, April 1990.

[Ste96] R. Steinmetz. Human perception of jitter and media synchronization. *IEEE Journal on Selected Areas in Communications*, 14(1):61–72, January 1996.

[Sti96] B. Stiller. *Quality-of-Service - Dienstgüte in Hochleistungsnetzen.* TAT, Nr.21, Int. Thomson Publication, Bonn, Germany, 1996.

[WR94] Th. Wahl and K. Rothermel. Representing Time in Multimedia Systems. In *Proc. Int. Conference on Multimedia Computing and Systems*, pages 538–543, Boston, MA, USA, May 1994.

[YH96] C.-C. Yuan and J.-H. Huang. A Multimedia Synchronization Model and Its Implementation on Transport Prptocols. *IEEE Journal on Selected Areas in Communications*, 14(1):212–225, January 1996.

[ZLS96] P. N. Zarros, M. J. Lee, and T. N. Saadawi. Interparticipant Synchronization in Real-Time Multimedia Conferencing Using Feedback. *IEEE/ACM Transactions on Networking*, 4(2):173–180, April 1996.

A Synchronization Parameters

The following parameters have been introduced within the document:

Table 1: Synchronization relevant parameters

parameter	meaning
B	buffer requirement in terms of number of PUs in a buffer
B_i	total buffer requirement for flow i
B_S	total buffer requirement for synchronization group S
d	delay
d_{min}	minimum delay
d_{max}	maximum delay
$d_{i,max}$	maximum delay of flow i
δ	$t_{p,0} - t_{s,0}$
δ_c	clock drift
Δ	maximum accepted asynchrony
Δ_t	some time difference (is to be further specified in the specific formula)
J	jitter (network and end system performance)
l	length (in bits) of a PU
ω	period of periodic PUs
$\underline{\omega}$	minimal difference of capture/presentation time between two successive PUs
p_a	probability of in time arrival of a PU
$p(d_l)$	probability of a delay smaller or equal than d_l
$p(b_{err})$	probability of a bit error in a PU
p_{err}	maximum synchronization error probability specified by the application
p_s	probability of synchronization errors
p_g	probability of offended guarantees in the system
$PU_{i,o}$	first PU of data flow i
t_P	a time point
t_r	a reference time
$t_{r,i}$	reference time of PU i
$t_{i,r,j}$	reference time of PU j in flow i
t_p	a presentation time
$t_{p,i}$	presentation time of PU i
$t_{i,p,j}$	presentation time of PU i in flow j
t_s	a sending time
$t_{s,i}$	sending time of PU i
$t_{i,s,j}$	sending time of PU i in flow j
t_{rc}	time, the PU was received in the receiver
$t_{rc,i}$	time, the PU i was received in the receiver
$t_{i,rc,j}$	time, the PU i of flow j was received in the receiver
t_n	time difference between first and n-th PU in the media schedule

B Summary of the Related Work

Table 2: Summary of the related work

specification of time					
[Ham72], [All91], [All83], [WR94]					
general overview					
[Ste90], 4-layer model: [BS96], [ME93], [SN95]					
synchronization					
	specification	synchronization	framework work	event-based	content-based
[Lit93]	TPN	play-out scheduling	defined		
[LG90]	OCPN	object retrieval and presentation algorithm			
[QWG93]	XOCPN	algorithm defined	defined		
[YH96]	RTSM	MSTP			
[ASCR95], [ASC96]	IPN				
[dCCdSSC92]	based on set theory				
[RH96]	media clocks				
[BHLM92]	reference points	object model, loose synchronization			
[GBB96]		server array, 2-phase startup protocol			
[IT95]		master/slave concept			
[RR93]		feedback technique, 4 policies			
[RH95]		ASP, master/slave concept, 2 policies			
[SSNA95]		Concord algorithm			
[Sre94]		master/slave concept	defined	eventually	
[ZLS96]		feedback technique			
[MB95]		for MHEG standard			
[PBCR95], [BCP+96]				yes	
[OKR95]					yes
requirements					
[Ste96]					
classifications					
[BS96], [ME93], [SN95], [SSF95], [PLL96], [dCCdSSC92]					
system model					
[SBC+97], [Bam97]					

Index