Analysis of the Concord Algorithm and the Adaptive Synchronization Protocol Using QoSy

Christina Class

February 23, 1998

Abstract

In this report the Concord synchronization algorithm and the Adaptive Synchronization Protocol are analyzed using Quality of Synchronization (QoSy). The concept of QoSy comprises the four parameters asynchrony, synchronization error probability, synchronization delays and buffer requirements which are introduced in this report. The analyses are described in detail and the results are presented.

1 Introduction

With the increasing development in computer engineering and technology leading to increasing computing performance, numerous *multimedia applications* like Video On Demand, Videoconferencing, Virtual Museums, and Teleeducation became feasible. These applications are characterized by the integrated processing of different media types, such as audio and video. Some of these media are time dependent and change their values as a function of time. For this reason, the recording, storing, and presentation of these data must take into account the data's time component. The task of multimedia *synchronization* is to retain data's temporal relationships. This task is aggravated in case of distributed multimedia applications.

Multimedia applications need to provide a minimum quality to be accepted by users. The quality must be provided by end-systems, the network as well as the application itself.

Therefore, the concept of Quality of Service allows for the specification of the quality provided by a system or required by the user. Within the literature many approaches and a lot of different synchronization algorithm exist. Different networks with different characteristics can be found in the field of multimedia applications as well. There is a need to relate these characteristics of the network and end-systems with characteristics of synchronization to determine, if there exists a synchronization method that can, within a given environment, provide the required quality to the user. In response to this need synchronization quality is defined comprising asynchrony, buffer requirements, synchronization error probabilities, and synchronization delay.

In this report a first approach to synchronization quality in a unicast environment is presented. Parameters are defined and an example analysis is presented. Following the definition of relevant terms in Chapter 2 a notion to refer to different time systems is introduced. Chapter 3 is devoted to specify the 4 characteristics of synchronization quality. Chapter 4 introduces the concept of quality of synchronization to analyze synchronization algorithms. Based on these quality characteristics, the Concord algorithm [15] is analyzed in Chapter 5 and the Adaptive Synchronization Protocol [13] is analyzed in Chapter 6. While Chapter 7 summarizes the analyses, Chapter 8 concludes the work and offers ideas to future work.

2 Analytical Environment

Based on definitions and a designed global time system, basic assumptions are made and a brief overview to related work is discussed.

2.1 Definitions

One *presentation unit (PU)* refers to the atomic information of a media stream, i.e. to the smallest information unit of the media stream that can be displayed independently. Examples of presentation units are video frames or audio samples. PUs are independent, i.e. no PU carries information relevant to others. In compression schemes like MPEG, compression is achieved by creating dependencies between PUs. To overcome these dependencies, the bit error probability has to be increased such as to cover also the probability that PUs cannot be presented due to a bit error in other PUs. Independent PUs are modeled by an adequate bit error probability.

Multimedia applications are characterized by the integrated processing of different media types. These media types may be *time dependent* or *time independent*. Time independent data, sometimes referred to as discrete media, does not have a time component. Time dependent data, in contrast, only is valid, if specified time conditions are fulfilled. Such a time condition may be the position of a video frame within a video sequence. If the frame is not displayed at the right time within the sequence, its information is not useful, i.e. not valid. The *valid time interval* is the interval in which a time dependent PU is valid. If, e.g., a PU corresponds to an audio sample of the length of 125 μs, the valid time interval $[x, y]$ means that a valid presentation time of the PU starts within $[x, y]$ and ends within $[x + 125\,\mu s, y + 125\,\mu s]$. The duration for the presentation of this single PU (125 μs) is not influenced by the valid time interval and vice versa.

2.2 Design of a Time System

For analyzing synchronization in distributed unicast applications, three different time systems need to be taken into account:

- The time system in the sender. All time points in the sender are denoted with t^s.
- The time system in the receiver. All time points in the receiver are denoted with t^r.
- The global time system. It serves as a reference time system and indicates the global, objective time. Time points in this time system are denoted with t^g.

The global time system is introduced to allow for the definition of synchronized clocks. The local time system of sender and receiver can be understood as a mapping of the global reference time system. This mapping is done specifically for each computer system, as in the real world, the local time systems of any two computers are not exactly the same. The mapping can be denoted as $M_r(t)$ in the receiver and $M_s(t)$ in the sender. For every machine we can define the inverse time mapping that computes the global reference time for the local time.[1] For the sender and receiver, this mapping can be denoted as $M_s^{-1}(t)$ and $M_r^{-1}(t)$.

If both clocks of sender and receiver are perfectly synchronized, we determine

$$\forall i : M_r^{-1}(t_i^r) = M_s^{-1}(t_i^s) \tag{1}$$

[1]This inverse mapping exists only (mathematically), if the time mapping is strictly monotonous. The assumption of a monotonous time mapping also in case of clock synchronization is reasonable as a non-monotonous time system may lead to severe problems in data bases and file systems.

or

$$\forall i : M_r(t_i^g) = M_s(t_i^g) \quad (2)$$

i.e. the time in both time systems is always the same. Real-world clocks are, in general, not synchronized perfectly, so the time in two time systems is different. This difference involves two aspects. Firstly, the time indicated at one specific time point may have different values. If the difference is constant, we can easily map from one time system to the other. Secondly, the clocks may drift. Drifting clocks advance with a different speed. This *clock drift* may have a constant value which results in one clock that is advancing the more of the other clock the more time passes by. It may also vary over time. In general, clocks are drifting, but as they are regularly re-synchronized to a global time, the drift is bounded by a minimum value and a maximum value. These bounds can be expressed as:

$$\forall i : b_{min} \leq |M_r^{-1}(t_i^r) - M_s^{-1}(t_i^s)| \leq b_{max}. \quad (3)$$

Often b_{min} is assumed to equal 0.

To ensure the correct display of PUs within their valid time interval, a mapping between the time system in the sender and the time system in the receiver has to be performed:

$$M_{s,r} : t^s \rightarrow t^r. \quad (4)$$

2.3 Assumptions

The *media schedule* specifies the correct play-out time for each PU. This time is called *media time*. The first PU of an application is assigned the media time 0. In applications handling stored data the media schedule must be specified. Different approaches can be used, e.g., the *object composition petri net* (OCPN) [7], the *inter-operable petri net* (IPN) [1], the *real-time synchronization model* (RTSM) [18], or *media clocks* [12].

In this report, we assume that PUs are sent according to the media schedule.[2] The accepted delay of a synchronization method determines PUs that have to be stored due to early arrival as well as PUs that have to be thrown away due to late arrival. This delay influences the probability of a presentation gap and, thus, the synchronization error probability. Capturing live data and sending them instantaneously to the receiver results in data being sent according the media schedule, as the media schedule in live data is implicitly given by the time, the data was captured. In case of stored data, the media schedule has to be explicitly given. The media schedule has to be read and interpreted on the sender's side and the data must be sent to the receiver according to the media schedule. To do so, a local synchronization mechanism has to be performed in the sender, controlling the synchronized sending of the data.

Each PU i has different times assigned to:

- The *sending time* is specified in the sender's time system and denotes the (local) time, the PU is sent ($t_{s,i}^s$).
- The *receiving time* specifies the time point in the receiver. It is given in the receiver's time system ($t_{r,i}^r$).
- The *presentation time* is the time the PU has to be presented. It is the target time of the synchronization and specifies the presentation time achieved by the synchronization method. It is specified in the receiver's time system and denoted as ($t_{p,i}^r$).[3]

[2] This restriction shall be dropped in future work to allow for a more general definition of the quality criterion asynchrony. But it was used as the base for the approach as described.

[3] If some constant time t_x^r is needed in, e.g., a device to prepare data for presentation (e.g., to decompress data), the presentation has to be triggered at $t_{p,i}^r - t_x^r$. This can be substituted by a new presentation time. If t_x^r varies, the variation results in additional asynchrony.

In case of different data streams that have to be distinguished, the indicator of the stream stands as first index, e.g., $t^r_{l,p,i}$ or $t^s_{k,s,j+1}$.

Finally, one *synchronization group* encompasses all flows or streams of an application that have to be synchronized. These streams do not have be sent necessarily from the same sender. This definition is similar to the one provided in [13].

2.4 Related Work

Synchronization is widely used in the literature. Synchronization consists of two steps to be fulfilled. Synchronization has to be specified, i. e., it has to be defined how data has to be played synchronously. Many approaches have been proposed, e. g., the object composition Petri Net (OCPN) [7], the real-time synchronization model (RTSM) [18] and the inter-operable Petri Nets (IPN) [1]. [11] defines a temporal reference framework allowing for the classification of specification schemes.

The second step of synchronization is to retain the specified time relationships. Many different approaches are proposed in the literature for content-based [9], event-based [10], and temporal synchronization [18], [19]. The two synchronization algorithms [13] and [15] that are analyzed in this paper are temporal synchronization approaches.

There is a need for a framework or a classification scheme due to numerous synchronization approaches. [2], [14], and [4] proposed initial frameworks.

Quality-of-Service (QoS) has been introduced as a powerful concept to define, monitor, and supervise the quality of communications. In particular, QoS is defined in Open Distributed Processing as "a set of quality requirements on the collective behavior of one or more objects" [6]. Slightly different definitions for QoS exist, e.g. within [8]. The use of QoS parameters to determine general synchronization issues on an algorithmic level has been mentioned in [20], synchronization definitions in terms of skew and jitter is included in [17], however, a detailed specification and utilization of generic synchronization quality parameters has not been reported yet. Therefore, this paper specifies and applies four generic QoS parameters for defining properties of synchronization algorithms.

3 QoSy: Quality of Synchronization

The concept of Quality of Service (QoS) is extended by parameters indicating the *Quality of Synchronization (QoSy)*. These parameters encompass asynchrony, synchronization error probability, synchronization delays, and synchronization buffer requirements [3].

Asynchrony

The stringent asynchrony measures the allowed deviation of the time interval between two subsequent presentations of PUs from the time interval in case of optimal presentation of these two PUs.[4] The time points given in the local time systems of sender and receiver are mapped to the global time by the inverse mapping function.

Definition 1: Δ_{ia} is the *maximum asynchrony in case of intra-stream synchronization*, if

$$\forall i \quad : \quad \max(0, M_s^{-1}(t^s_{s,i+1}) - M_s^{-1}(t^s_{s,i}) - \Delta_{ia})$$

[4] Note that a time interval can be of length zero to cover inter-stream asynchrony.

$$\begin{aligned} &\leq M_r^{-1}(t^r_{p,i+1}) - M_r^{-1}(t^r_{p,i}) \\ &\leq M_s^{-1}(t^s_{s,i+1}) - M_s^{-1}(t^s_{s,i}) + \Delta_{ia}. \end{aligned} \tag{5}$$

The term $\max(0, M_s^{-1}(t^s_{s,i+1}) - M_s^{-1}(t^s_{s,i}) - \Delta_{ia})$ avoids a large term Δ_{ia} allowing for the violation of the ordering aspect of intra-stream synchronization.

The following definition of inter-stream asynchrony is a general one, as it allows for any two streams to be sent from two different senders.

Definition 2: Δ_{ie} is the *maximum asynchrony in case of inter-stream synchronization* for the two flows k and l, if

$$\begin{aligned} \forall i \quad : \quad & M_{s_1}^{-1}(t^{s_1}_{k,s,i}) - M_{s_2}^{-1}(t^{s_2}_{l,s,i}) - \Delta_{ie} \\ \leq \quad & M_r^{-1}(t^r_{k,p,i}) - M_r^{-1}(t^r_{l,p,i}) \\ \leq \quad & M_{s_1}^{-1}(t^{s_1}_{k,s,i}) - M_{s_2}^{-1}(t^{s_2}_{l,s,i}) + \Delta_{ie}. \end{aligned} \tag{6}$$

Synchronization Delays

The delay of data that is transmitted live, e.g., during a video conference, as well as the delay before the first data is presented to the user, may be critical for the quality perceived by the user. Synchronization may increase these delays by buffering data to smooth the jitter. For this reason, the delay is an essential part of synchronization quality.

Buffer Requirements

Whenever the network delay can vary, synchronization needs to buffer PUs arriving too early to ensure an in-time play-out of PUs. These *buffer requirements* can serve as an indication of the cost of synchronization algorithms, as by buffering unreliabilities of the underlying infrastructure are smoothed.

Synchronization Error Probability

Three different kinds of synchronization errors are defines [3]:

1. There exist one or more consecutive valid time intervals of PUs in which the corresponding PU is not displayed (*gap*).[5] Reasons for a gap are late arriving PUs, losses of PUs, or unreliable end-systems.

2. One or more PUs are displayed outside their valid time intervals. This is referred to as *loss of synchronization.*

3. The mapping $M_{s,r}$ is incorrect or the time systems t^s and t^r are not mapped at all. In this case a *loss of time* occurs.

The second types of synchronization errors may occur due to synchronization information being modified by an unperceived bit error, due to errors in the synchronization function or due to additional delays in the end-system. If the synchronization method

[5] Note that this definition of a gap defers from the definition in [16].

is implemented correctly, these errors only occur due to problems in the end-system and are, therefore, not included in the analysis. For the same reason, an incorrect time mapping resulting in the third type of synchronization errors, is not included in the analysis of synchronization quality. The parameter *synchronization error probability* is, thus, for analysis purposes restricted to the case in which a PU is not presented in correct time or is not presented at all.

The synchronization error probability can serve as an indicator for the quality of the synchronization scheme. The parameter synchronization error probability is not very straightforward, but is a very important factor to determine synchronization quality. The delay of the synchronized data is a relevant quality factor, as delay may be very disturbing in real-time applications. It is possible to synchronize data with a very low delay, but in this case all data arriving too late must be thrown away. This leads to an increased value for the synchronization error probability which expresses the low total quality of the scheme.

It is not easy to define a synchronization error for an entire synchronization group as there exist multiple possibilities. The interpretation of the synchronization error is depending on the aim of the analysis and the error must be clearly defined in every analysis. Some of these possible definitions of a synchronization error for a synchronization group are:

- Following the master/slave concept: a synchronization error is given, if the master PU cannot be displayed in time.
- Following the master/slave concept: a synchronization error is given, if the master or more than 50 % of the slave PUs cannot be presented in time.
- A synchronization error is given, if any of the PUs cannot be presented in time.

4 The Analyses of Synchronization Algorithms Applying QoSy Parameters

4.1 Model

In distributed multimedia applications five different components are distinguished for the purpose of analyzing synchronization algorithms as presented in Fig. 1:

1. Application,
2. User,
3. Synchronization algorithm,
4. End-system, and
5. Communication system.

The application has data that is, at least a part of it, time dependent and, thus, requires mechanisms to ensure a synchronized play-out of the data. The synchronized play-out is defined with reference to a valid time interval or a given synchronization specification. The data itself has characteristics being relevant to the synchronization, e. g., the size of one presentation unit, the play-out time of one presentation unit, and the periodicity.

On one hand, the user of an application has requirements with respect to the entire application as well as to the synchronization itself. Possible requirements to the synchronization, as the maximum accepted skew and jitter, may be found in [17]. Application requirements that are relevant to the synchronization algorithm include, e. g., the data format and cost requirements in form of buffer requirements.

On the other hand, there exists an underlying end-system and the communication system. The end-system has characteristics influencing the synchronization. One important characteristic is the local time system and how well it is synchronized to the local time system of the sender. Another important criterion is an end-system's support of real-time requests. The synchronization algorithm initiates the synchronous play-out of data. Synchronization cannot be guaranteed, if the end-system does not play out the data instantaneously or with a specified delay that has be taken into account by the synchronization mechanism.

The communication system is specified by the characteristics of the underlying network. Data that is to be played-out, but did not (yet) reach the end-system, leads to a synchronization error. The delay as well as the packet loss rate of the underlying network play an important role. The network and the end-system offer service with given characteristics. In general, this service does not allow the application to play out data synchronously, especially if there exist delay variances in transmission. For this reason, synchronization methods have to be applied to fill the gap between the requirements of applications and users and the "service" characteristics of the end-system and the network.

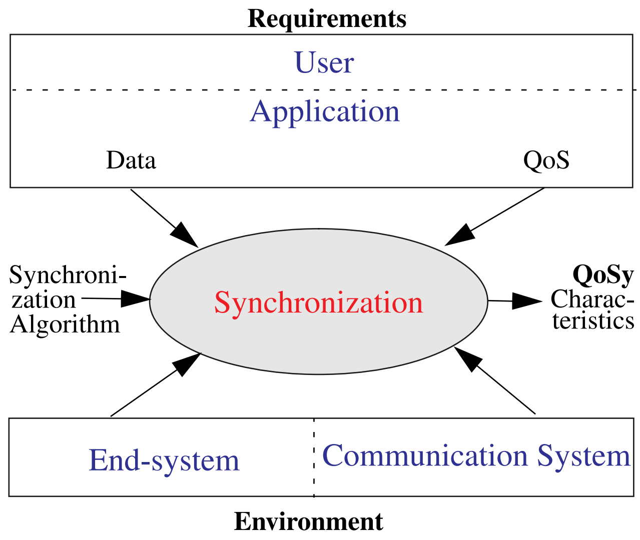

Figure 1: The Analysis Model of Distributed Synchronization

4.2 Analyzing Synchronization Algorithms

Applying the model above and the Quality of Synchronization parameters as defined in Chapter 3 synchronization algorithms can be analyzed. The goal of this analysis is to determine values for QoSy parameters describing the algorithm independently from end-systems, the communication system, the application, and the user. For this reason, the characteristics of these dependent systems have to be parameterized.

Analyzing an algorithm is a demanding task which is not supported by rules or tools. However, clearly separated parameters allow for the direct calculation of the asynchrony. In a first step algorithms have to be analyzed assuming perfectly synchronized clocks and the simplest case as defined in the algorithm. Afterwards, step by step all factors are taken into account that aggravate the synchronization by determining the changes to the parameters introduced by these factors. Especially the clear definition of the different time systems made them usable for this analysis (cf. Chapter 2.2).

The analysis remains independent from the underlying network as well as from the end-systems. Data characteristics are also encompassed into the analysis as parameters. This allows for the calculation of QoSy parameters for a given environment and application, if their characteristics are specified. The result can be used to determine, if a given infrastructure and a given synchronization method are suitable to provide requirements demanded by the user and the application.

For this analysis two synchronization methods belonging to two different classes of synchronization have been chosen. The *Concord* algorithm [15] belongs to the class of rigid synchronization algorithms whereas the *Adaptive Synchronization Protocol* [13] defines an adaptive synchronization algorithm.

Both algorithms were analyzed by parameterizing the environment characteristics. The main focus in the analysis was given to some high level results that might be suitable for comparison. Especially the asynchrony as characteristic was focused. In the analysis we concentrated on an application, where data comes from one or multiple senders and is to be displayed in one receiver. We made the assumption that data is sent live, i. e. effects resulting from exploiting the fact that data is stored and may be sent in advance are ignored. Also we assumed that all data streams are present from the beginning of the application, i. e. that no data stream is joining late in the application.

Using synchronization quality to assess synchronization schemes, QoSy parameters are referred to as guaranteed parameters. For this reason, the analysis has to take into account the worst case.

5 The Concord Algorithm

The analysis of the Concord Algorithm is based on [15]. The algorithm is founded on the principle "if every packet is delayed in the buffer such that is suffers cumulatively (in the network and buffer) a delay equal to the maximum network delay, the receiver can reproduce a jitter-free playback"[15]. The playback of data is perfectly synchronized achieving jitter-free playback.

The Concord algorithm is an easy-to-understand and easy-to-implement algorithm as it achieves synchronization by guaranteeing for a constant delay of packets for each data stream. As data is sent periodically, the task of synchronization is simplified and the influence of unknown network delays only plays a role with respect to the first data packet.

The algorithm computes the delay each packet has to suffer and operates a synchronization buffer that is implemented as a shift register. To compute the total end-to-end delay following values are required: the *maximum acceptable delay* (MAD), the *maximum packet loss rate* (PLR), and the *packet delay distribution* (PDD). The knowledge of the *cumulative distribution function* (Cdf) of PDD is assumed. A *total end-to-end delay* ($TED = \Delta$) is calculated which is constant for every packet p and consists of the *network delay* (nd_p) and the *buffer delay* (bd_p). Three policies can be chosen to set the $TED = \Delta$ of packets: to minimize the PLR, to minimize the Δ, or to minimize some sort of hybrid costs. Of course, a "Δ can be chosen satisfying both the MAD and PLR constraints *iff* $(1 - Cdf(MAD)) \leq PLR$"[15].

[15] assumes that packets are sent periodically with the period T_r. As the packets are also to be displayed periodically, we assume that one packet refers to one PU and that PUs are periodic. T_r denotes the time during which a PU is presented and valid.

5.1 Asynchrony

5.1.1 Basic Algorithm

The simplest case for the Concord synchronization algorithm is given, if:

- clocks in sender and receiver do not suffer any clock drift, and if
- the network delay nd_0 of the first packet is known.[6]

The play-out time of PU i in the receiver's time system is specified by

$$t^r_{p,i} = i \cdot T^r_r + t^r_{r,0} + \Delta^r - nd_0. \tag{7}$$

As nd_0 is the delay of the first packet and as it is exactly known, $t^r_{p,i}$ can be mapped to the sender based on the assumption of no clock drift:

$$t^s_{p,i} = i \cdot T^s_r + t^s_{s,0} + \Delta^s. \tag{8}$$

As Δ^s and $t^s_{s,0}$ are constant for all i and as there is no clock drift, the play-out of data is perfectly synchronized with a play-out rate of T^s_r and T^r_r respectively.

5.1.2 Algorithm for Unknown Network Delay

The network delay of the first packet (or any other packet) can be known in case of synchronized clocks, as it can be derived by time-stamps. In [15] an algorithm is proposed in case of an unknown network delay for the first packet or in case the network delay cannot be derived.

The approach considered uses a *special control packet* as a first packet to be sent (it is ignored for presentation purpose). It is assumed that the network can guarantee the packet delivery within $[nd_{min}, nd_{min} + \lambda]$.

First case $\lambda \leq T_r$: The first packet is inserted in the final slot of the synchronization buffer and, thus, bd^r_0 is maximum. Let the actual network delay be $nd_{min} + \delta$ with $0 \leq \delta \leq \lambda$ and $0 \leq \lambda \leq T^r_r$. The first packet arrives with a delay of $nd_{min} + \delta$ and is delayed bd_{max} in the synchronization buffer. The optimal total end-to-end delay is $\Delta_{opt} = nd_{min} + bd_{max}$ and, thus, the end-to-end delay in the described case is $\Delta = \Delta_{opt} + \delta = nd_{min} + \delta + bd_{max}$. The play-out for packets is determined by
$t^r_{p,i} = i \cdot T^r_r + t^r_{r,0} + \Delta^r - nd_0 = i \cdot T^r_r + t^r_{r,0} + \Delta_{opt} + \delta - nd_0$
$= i \cdot T^r_r + t^r_{r,0} + \Delta_{opt} + \delta - nd_{min} - \delta = i \cdot T^r_r + t^r_{r,0} + bd_{max}$.

[6] Note, that in the following formalization the first packet is assigned number 0. [15] defines number 1 for the same packet.

As $bd_{max} + \delta = bd_0 = \Delta_{opt} - nd_0 + \delta$ holds, the play-out is perfectly synchronized with a playback rate of T_r and is delayed by δ compared to the optimal play-out in case of known network delay of the first packet.

Second case $\lambda > T_r$: In this case the uncertainty of the delay the first packet suffered is more than the time interval between the presentation of two subsequent PUs. The Concord algorithm puts the first packet in the final slot of the synchronization buffer and delays it for bd_{max}. For this reason, it is possible that packets arriving with a little delay cannot be stored in the buffer, as there are not enough slots available. In this case, the packets in the buffer are shifted to the right side, leading to the first packet(s) in the buffer being dropped.

The initial total end-to-end delay is bounded by $\Delta_{opt}+\lambda$. For the analysis of asynchrony the worst case has to be investigated. The worst case is given, if packets arrive with a delay long enough to avoid buffer overflow and a packet arrives that suffered from the minimum delay. The number of packets thrown away is given by $\left\lfloor \frac{\lambda}{T_r} \right\rfloor$. The new TED is specified by $\Delta_{end} = \Delta_{opt} + \delta$ with $\delta = \lambda - \left\lfloor \frac{\lambda}{T_r} \right\rfloor \cdot T_r$. For this reason, the maximum asynchrony between two subsequent packets is bounded by $\left\lfloor \frac{\lambda}{T_r} \right\rfloor$ packets or by $\left\lfloor \frac{\lambda}{T_r} \right\rfloor \cdot T_r$. This is valid for one single data stream. The maximum asynchrony between packets of two different data streams is specified by the maximum of $\left\lfloor \frac{\lambda}{\tau_r} \right\rfloor$ of both streams.

5.1.3 Clock Offset and Drift

The Concord algorithm uses time-stamps to determine the delay of the first packet. In case of not perfectly synchronized clocks, the time stamp of the packet has a different meaning for sender and receiver. Assuming that clocks are regularly re-synchronized and that the asynchrony of the clocks is bounded by ϵ, specified as a value in the global (objective) time system, the real network delay $nd_{real} = M_r^{-1}(nd_0^r)$ of the first packet is within the interval of $[M_r^{-1}(nd_0^r) - \epsilon, M_r^{-1}(nd_0^r) + \epsilon]$.

The Δ^g at which the algorithm is operating, i.e. $\Delta^g = M_r^{-1}(\Delta^r)$ is within the interval of $[\Delta_{opt}^r - \epsilon, \Delta_{opt}^r + \epsilon]$. To ensure that the PLR is maintained, Δ^r is chosen as $\Delta^r = \Delta_{opt}^r + \epsilon$ and $nd_{0,new}^r = nd_0^r - \epsilon$.

Packets are sent from the sender's side with T_r^s and played out at the receiver's side with T_r^r which may vary over time (due to clock re-synchronization adjusting the clock drift) and may differ from T_r^s by ϵ. The worst case is determined for packet i where the offset between both clocks is ϵ and clocks are perfectly synchronized during the play-out of packet $i+1$. In this case the additional asynchrony in the play-out of packet i and packet $i+1$ due to clock offset and drift is given by ϵ. If the clock offset is constant over time, i.e. the two clocks are not drifting, there is no additional asynchrony induced by the clock offset. In this case $M_r^{-1}(T_r^r) = M_s^{-1}(T_r^s)$ holds.

5.1.4 Drifting Production and Service Rate

For analysis purpose it is assumed that sender and receiver drifts do not occur.

Source faster than the receiver: Let $\tau_r = M_r^{-1}(T_r^r)$ and $\tau_s = M_s^{-1}(T_r^s)$. In the case of a source being faster than the receiver, there is a risk of buffer overflow in the receiver, as the production rate is higher than the service rate, i.e. the playout period in the source is smaller, i.e. $\tau_s < \tau_r$. The analysis in [15] assumes 0 as a value for nd_0.

The idea of the Concord algorithm is to throw away packets on a regular basis to avoid buffer overflow. We calculate the maximum number of packets between two packets that have to be thrown away. A buffer overflow will occur, whenever a packet arrives, while the buffer is full. Packets are sent with the service rate and are played out with the production rate. If the service rate is higher than the production rate, a buffer overflow occurs in case of every i packets with $i = \frac{\tau_r}{\tau_r - \tau_s}$. Each packet is displayed at $t_{p,0} + i \cdot \tau_r$, i.e. with a delay of $\tau_r - \tau_s$ compared to its normal play-out in reference to the preceding packet. After i packets the cumulated delay of play-out is as big as the period between the play-out of two subsequent PUs. For this reason, one packet may arrive in the buffer (if it has minimal delay), while the first packet in the buffer has not yet been serviced and the buffer is full. For this reason, at least every i-th packet has to be thrown away. Assuming that the service rate in the sender is correct (cf. Chapter 2) we can state that each packet in the sender is played out with an asynchrony out of $[0, \tau_r - \tau_r/(\frac{\tau_r}{\tau_r - \tau_s})] = [0, \tau_s]$ as a packet is dropped, when the asynchrony reaches τ_r and the asynchrony is then reset to 0 again. Therefore, the maximum asynchrony is bounded by τ_s.[7]

Source slower than the receiver: The receiver services packets at a faster rate than the server sends them. Analogously, a buffer underflow will occur for every i packets with $i = \frac{\tau_r}{\tau_s - \tau_r}$. For this reason, every i-th packet is duplicated[8]. The asynchrony of presented packets compared to correctly sent packets is within the interval of $[0, \tau_r - \tau_r/(\frac{\tau_r}{\tau_s - \tau_r})] = [0, 2 \cdot \tau_r - \tau_s]$. Thus, the asynchrony is bounded by $2 \cdot \tau_r - \tau_s$. It is concluded that the asynchrony in case of drifting production and service rate is bounded by $2 \cdot \tau_r - \tau_s$.

5.1.5 Multiple Stream Extension

The multiple stream extension in [15] maps the proposed algorithm to the case of inter-stream synchronization. This is done by calculating the TED for streams with respect to all characteristics of streams and their links of the application.

For analysis purposes it is assumed here that all streams are of the same importance (no master/slave concept) and that no additional features, as redundant streams, are available. In this case the asynchrony of inter-stream synchronization between two streams can be mapped to the maximum intra-stream asynchrony of the two streams. For this reason, the maximum asynchrony of a synchronization group is specified by the maximum intra-stream asynchrony of the streams in the group.

5.2 Synchronization Error Probability and Maximum Delay

[15] assumes a known packet delay distribution of the underlying network. It is assumed, that the user specifies the *packet loss rate*, being acceptable. Therefore, the *maximum acceptable delay (MAD)* is defined to guarantee the user specified packet loss rate. This loss rate is the part of synchronization errors being independent of bit errors in the underlying network. Also, the *total end-to-end delay (TED)* may be specified based on the packet loss rate. The TED is equal to the maximum delay occuring in the network

[7] This asynchrony may be reduced by throwing away additional packets, however leading to an increased synchronization error.

[8] As audio might not be duplicated, a presentation pause may be made.

and influences the packet loss rate. The smaller the total end-to-end delay, the more packets arrive late, increasing the packet loss rate as these packets are dropped.

5.3 Buffer Requirements

Buffer requirements per stream i are given by $\left\lceil \frac{\Delta_i - nd_{min,i}}{T_{i,r}} \right\rceil$ in number of packets and $\left\lceil \frac{\Delta_i - nd_{min,i}}{T_{i,r}} \right\rceil \cdot l(p_i)$, where $l(p_i)$ denotes the length of a PU in stream i[9]. Thus, buffer requirements in the entire application are determined by $\sum_i \left\lceil \frac{\Delta_i - min_i}{T_{i,r}} \right\rceil \cdot l(p_i)$

5.4 QoSy Parameter Values for the Synchronization Group

QoSy parameter values for one synchronization group being synchronized with the Concord algorithm can be derived, if all parameter values for the streams are given.

5.4.1 Asynchrony

Based on the assumption that all data streams are present from the start of the application, and that the total end-to-end delay is the same for all data streams [15] the inter-stream asynchrony is 0 in case of perfect clocks and bounded to ϵ in case of bounded clocks. According to [3] the total asynchrony is specified by the maximum intra-stream asynchrony of all streams plus 0 or ϵ in case of perfectly synchronized clocks or bounded clocks respectively.

5.4.2 Delay

The delay for the synchronization group results in the maximum delay of all streams, as this is chosen for the delay of all streams (compare [15]).

5.4.3 Synchronization Error

To calculate the synchronization error of a synchronization group, a definition of a synchronization error of a group must be provided. A synchronization error is present, if any one of the PUs cannot be displayed in time. For this reason, the synchronization error probability for the synchronization group is given by

$$1 - \prod_i (1 - p_i(err)), \tag{9}$$

where i denotes all streams in the synchronization group.

5.4.4 Buffer Requirements

Buffer requirements of the synchronization group are specified by the sum of the buffer requirements for each stream. Let Δ_g denote the delay for all streams, the buffer require-

[9] If buffer requirements need to be an upper bound, $l(p)$ denotes the maximum PU size. Since buffer requirements necessarily do not need to describe the upper bound, the mean size of PUs can be taken to define $l(p)$.

ment is specified by

$$\sum_i \left\lceil \frac{\Delta_g - \Delta_{i,min}}{\tau_{i,r}} \right\rceil. \tag{10}$$

6 The Adaptive Synchronization Protocol

The *Adaptive Synchronization Protocol (ASP)* as described in [13] is an adaptive synchronization algorithm with centralized control. In monitors the underlying network and adapts to changing network conditions. The control mechanism of this protocol is on the buffer level. One synchronization group consists of one controller for the entire group and one agent per stream. An agent is a software entity controlling an individual stream. ASP is based on stream rates, i.e. data is periodic. The controller sends a start message to source agents and sink agents start playing out of data after a delay of Δ. Δ is the maximum delay of all data transmissions plus the delay that is needed to fill buffers to a minimum level.

Normally the production rate is equal to the release rate. In case of any problems (buffer under- or overflow) in sink agent(s), the release rate is adapted. This does not influence the production rate. As the ASP tries to control the delay by controlling the play-out buffer, there is an implicit assumption in [13] that the packet sequence must not be changed by the underlying network.

The ASP algorithm consists of four different protocols. The goal of the algorithm is to minimize the end-to-end delay or to minimize the loss probability, respectively, depending on the synchronization policy chosen and the actual network state. To achieve this goal, in each play-out buffer two relevant marks are defined: the *low water mark (LWM)* describing the minimum buffer fill acceptable and the *high water mark (HWM)* describing an upper limit for the buffer fill (cf. Fig. 2). Both water marks define the accepted jitter, whereas the delay that is the base for jitter calculation (the jitter is the difference between maximum and minimum delay) may be adapted to the network condition during an application. This makes the synchronization algorithm an adaptive one.

There is always one master stream and multiple slave streams. The buffer of the master stream has two additional relevant values. The *lower target boundary (LTB)* and the *upper target boundary (HTB)* mark define a target buffer (cf. Fig. 2). The ASP aims to keep the end-to-end delay of the application within the target area of the master. The master observes the actual delay by observing the actual buffer fill $d_B(t)$. To avoid the master entering an adaption phase due to a momentary problem, the actual buffer fill is smoothed over time. [13] propose a geometric smoothing function

$$d_B(t_i) = \alpha \cdot d_B(t_{i-1}) + (1-\alpha) \cdot \text{ActBufferDelay}(t). \tag{11}$$

If a slave stream observes any problem, i.e. its buffer risks a buffer over- or underflow, the slave stream may ask to become a master. If a master starts an adaption phase, a time is set, when the adaption phase is finished (t_e) and a new play-out rate R_2^A is calculated with

$$R_2^A = R_N \cdot (1 + R_{corr}) \tag{12}$$

and

$$R_{corr} = (d_B(t) - (LTB + (UTB - LTB)/2)/L, \tag{13}$$

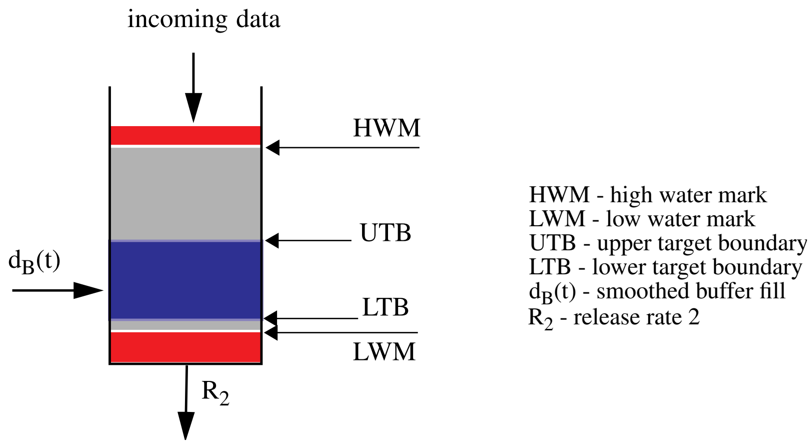

Figure 2: The Buffer in the Master Stream of ASP

where L is the length of the adaption phase and R_N is the normal play-out rate.

An *Adapt* message is sent to the slave streams, carrying the end-time of the adaption phase as well as the media time that should be valid by the end of the adaption phase. The new play-out rate of the slave is given by

$$R_s^A = R_N \cdot (M(t_e) - M(t_a))/(t_e - t_a), \tag{14}$$

where t_e is the time the adaption phase ends, t_a is the time the adaption phase starts, $M(t_a)$ is the current media time, and $M(t_e)$ is the media time that shall be valid at the end of the adaption phase.

In fact, ASP defines a class of protocols. The policy (the two policies *minimum delay* as well as *minimum loss* are described in detail) as well as the smoothing function for the buffer delay and the width of the target buffer influence the protocol behavior to a great extent. Therefore, one synchronization policy is analyzed in detail.

Depending on the synchronization policy, the target buffer areas may be chosen. [13] propose two different synchronization policies: (1) *minimum delay policy* and (2) *minimum loss policy*. The evaluation assumes (1) that the master stream and the slave streams remain stable and (2) that the roles change.

As in all synchronization algorithms, the buffer in ASP serves the purpose of smoothing jitter in the transmission delay. For this reason HWM corresponds to data that is delayed, until the maximum accepted delay is elapsed. LWM corresponds to data that is only delayed to a minimum end-to-end delay. In algorithms like the Concord algorithm, data arriving with a minimum delay is delayed for the whole "buffer size" and played-out afterwards. Assuming that ASP does not invert the order of presentation units, each presentation unit will be placed in the buffer according to its delay suffered from during transmission. If data is to be played out that is near the LWM, i. e. if the target buffer area is near to LWM, the data is to played out shortly after suffering from a minimal delay.

6.1 Asynchrony

Based on perfectly synchronized clocks, the normal play-out in ASP is at time: $t_0 + \Delta + TS(u)$, where Δ denotes the delay and $TS(u)$ denotes the time stamp of the presentation unit. There are two different cases to be distinguishes when a master enters an adaption phase:

1. **Speeding up adaption:** The smoothed buffer fill is above UTB, i. e., the play-out of the PUs has to be sped up, such that the buffer is emptied.
2. **Slowing down adaption:** The smoothed buffer fill is below LTB, i. e., there is a risk of buffer underflow in the master and the play-out of presentation units has to be slowed down.

The goal of both different adaption modes (speeding up and slowing down) is that the actual buffer fill $d_B(t)$ approximately equals $(UTB - LTB)/2 + LTB$.

6.1.1 Minimum Delay Policy

The minimum delay policy tries to find the minimum end-to-end delay. The intra-stream synchronization quality is assumed to be given [13]. The master stream is the one with the currently maximum transfer delay. The target area is defined as $LTB = LWM$ and $UTB = LWM + \omega$ with $\omega = c \cdot j_M$ and j_M denoting the jitter.

Single data units are played out at

$$t_0 + \Delta_{min} + (UTB + LTB)/2 + TS(u). \tag{15}$$

Δ_{min} is directly dependent on LWM and determines the minimum delay respecting the buffer in the master stream. As the stream suffering from the current maximum delay in the synchronization group becomes the master stream according to this synchronization policy, this delay corresponds to the current maximum delay in the synchronization group. That is, if there is no current adaption phase, PUs are presented at

$$t_0 + \Delta_{min} + \frac{1}{2} \cdot c \cdot (\Delta_{max} - \Delta_{min}) + TS(u) = t_0 + \Delta_{min} + \frac{1}{2} \cdot c \cdot j_M + TS(u). \tag{16}$$

For this reason each presentation unit suffers from a delay of

$$\Delta_{min} + \frac{1}{2} \cdot c \cdot (\Delta_{max} - \Delta_{min}) = \Delta_{min} + \frac{1}{2} \cdot c \cdot j_M. \tag{17}$$

c depends on the smoothing function of the buffer and describes how much of the jitter of the data stream is allowed to reflect as jitter of the buffer delay within the target area, $0 \leq c \leq 1$. [13] proposes $c = 0.3$.

A. Analysis without Master Switching

1. Intra-Stream Asynchrony in the Master Stream: If the master does not need to enter an adaption phase, the intra-stream asynchrony is 0.

Speeding up adaption: If the master enters an adaption phase, its smoothed buffer fill is higher than UTB, i.e., PUs arrive too early and suffer a transmission delay of $< \Delta_{min}$. The master enters the adaption phase and adjusts the minimum delay to $\Delta_{min'}$, with $\Delta_{min'} < \Delta_{min}$ such that the new minimum delay allows for the $d_B(t)$ to remain within the target buffer area with a high probability. The master enters the adaption phase as soon as $d_B(t)$ is observed to have left the target area. Assuming the worst case, the length of the adaption interval is very short, i. e., the length of this interval is chosen such that after the next $T_{r,M}$ passed, the adaption phase is finished.[10] For calculation purposes, it is assumed that the master starts an adaption phase just after displaying a PU. During one time interval $T_{r,M}$, the master needs to display

$$\left\lceil \frac{c \cdot j_M \cdot \frac{1}{2}}{T_{r,M}} \right\rceil + 1 \tag{18}$$

PUs as this corresponds to an upper bound for the number of PUs equivalent to $|d_B(t) - (LTB + (UTB - LTB)/2)|$. The play-out time of each of the PUs that are to be presented during the interval of length $T_{r,M}$ is

$$\frac{T_{r,M}}{\left\lceil \frac{c \cdot j_M \cdot \frac{1}{2}}{T_{r,M}} \right\rceil + 1}. \tag{19}$$

For this reason the additional asynchrony in the adaption phase, which is of minimal perceivable and reasonable length is

$$T_{r,M} - \frac{T_{r,M}}{\left\lceil \frac{c \cdot j_M \cdot \frac{1}{2}}{T_{r,M}} \right\rceil + 1} \tag{20}$$

Assuming a given length of the adaption phase, where L denotes this length in number of $T_{r,M}$, the play-out time for each of the

$$\left\lceil \frac{c \cdot j_M \cdot \frac{1}{2}}{T_{r,M}} \right\rceil + L \tag{21}$$

PUs during the interval of length $L_M \cdot T_{r,M}$ is

$$\frac{T_{r,M} \cdot L_M}{\left\lceil \frac{c \cdot j_M \cdot \frac{1}{2}}{T_{r,M}} \right\rceil + L_M}. \tag{22}$$

Therefore, the intra-stream asynchrony is given by

$$T_{r,M} - \frac{L_M \cdot T_{r,M}}{\left\lceil \frac{c \cdot j_M \cdot \frac{1}{2}}{T_{r,M}} \right\rceil + L_M}. \tag{23}$$

The maximum intra-stream asynchrony during the adaption phase is bounded by $T_{r,M}$ which is a true upper limit that will never be reached. The buffer is measured in time (units) and the adaption already reacts when $d_B(t)$ is out of range, which is the case before the time for the display of one PU is elapsed.

Slowing down adaption: If the master stream enters this adaption phase, it is in a risk of buffer underflow. For this reason, the play-out of PUs has to be slowed down by

$$\left\lceil \frac{c \cdot j_M \cdot \frac{1}{2}}{T_{r,M}} \right\rceil \tag{24}$$

[10] The length of the adaption phase is a relevant parameter for ASP and also a relevant parameter for the synchronization quality. However, [13] does not make any suggestions how to choose the length of the adaption phase.

PUs. To react as soon as possible, the play-out of the next PU has to be delayed by a time of

$$c \cdot j_M \cdot \frac{1}{2} \tag{25}$$

describing the intra-stream asynchrony. Assuming $L_M \cdot T_{r,M}$ to denote the length of the adaption phase, the intra-stream asynchrony is given by

$$\frac{c \cdot j_M \cdot \frac{1}{2}}{L_M}. \tag{26}$$

2. Intra-Stream Asynchrony in the Slave Stream: In case of no adaption phase going on, the intra-stream asynchrony in the slave is limited to 0. The delay of the PUs in the slave stream is dependent on the delay of the units in the master stream. Assuming a live application or an application in which the PUs are sent according to the media schedule, the delay of data is $\Delta_{min} + c \cdot j_M$ whereas Δ_{min}, c, and j_M are taken from the master stream.

Speeding up adaption: During the adaption phase two cases have to be distinguished. First, the adaption phase is already finished when the *Adapt* message reaches the client. In this case, the PUs must be skipped and the correct PU calculated with the new media time from the adaption phase must be played out. This results in a synchronization error (indirectly it determines a late arrival of PUs) and is not counted as asynchrony. In case the *Adapt* message reaches the slave agent before the adaption phase is finished, the maximum asynchrony for the intra-stream synchronization is bounded by $T_{r,S}$. The exact value is determined by

$$T_{r,S} - \frac{T_{r,S} \cdot \left\lceil \frac{l}{T_{r,S}} \right\rceil}{\left\lceil \frac{c \cdot j_M \cdot \frac{1}{2}}{T_{r,S}} \right\rceil + \left\lceil \frac{l}{T_{r,S}} \right\rceil}, \tag{27}$$

where c and j_M are the width of the target buffer and the jitter in the master stream and l is the time elapsed between the *Adapt* message and the end of the adaption phase.)

Slowing down adaption: If the master enters an adaption phase to slow down the play-out, three different cases have to be distinguished in the slave stream. First, the adaption phase in the master is delayed, until the slave receives the *Adapt* message. In this case the slave stream perceives an intra-stream asynchrony bounded to

$$\frac{c \cdot j_M \cdot \frac{1}{2}}{\left\lceil \frac{L_M \cdot T_{r,M}}{T_{r,S}} \right\rceil}, \tag{28}$$

as the delay of in total $c \cdot j_M \cdot \frac{1}{2}$ has to be achieved within $\left\lceil \frac{L_M \cdot T_{r,M}}{T_{r,S}} \right\rceil$ presentation intervals in the slave master stream.

Secondly, the *Adapt* message reaches the slave before the adaption phase is ended. In this case, the intra-stream asynchrony in the slave is bounded to

$$\frac{c \cdot j_M \cdot \frac{1}{2}}{\left\lceil \frac{l}{T_{r,S}} \right\rceil}, \tag{29}$$

whereas l denotes the length of the time interval between the reception of the *Adapt* message and the end of the adaption phase.

Thirdly, the slave agent may have played out one more presentation unit that are to be played out after the adaption phase ended. This case is interpreted as an synchronization error and does not count as asynchrony. The play-out is resumed with the correct presentation unit at the announced end of the adaption phase.

3. Inter-Stream Asynchrony: In case of normal playback (and perfectly synchronized clocks), the inter-stream asynchrony is 0. In case of an adaption phase, there are two different possibilities:

1. The master delays the adaption phase, until all slave streams got the *Adapt* message which is sent at once, when realizing a $d_B(t)$ outside the target area.

   Speeding up and slowing down adaption: In both cases the inter-stream synchronization is not influenced, as all streams are adapted within the same time interval. The worst case assumption for the intra-stream asynchrony as mentioned above does hold also. While waiting for the adaption phase to start, $d_B(t)$ might be farer away from the target buffer area than before. This does not influence the adaption as the *Adapt* message has been sent based on the master's data, at the time it was sent, and at this moment above worst case assumptions hold.

2. The master starts its adaption phase at once and its play-out is in advance from the play-out of the streams.

   Speeding up adaption: The inter-stream asynchrony for the master stream and one slave stream depends on the delay for the *Adapt* message. Also the inter-stream asynchrony between different slave streams depends on the delays for the *Adapt* messages for each slave stream. To become independent from these delays the upper bound of the inter-stream asynchrony is given by[11]

   $$\frac{1}{2} \cdot j_M \cdot c, \tag{30}$$

   as this is an expression in time. This asynchrony also covers the case that one *Adapt* message reaches the slave after the adaption phase ended.

   Slowing down adaption: In case the play-out of further presentation units by the slave is regarded as a synchronization error, the inter-stream asynchrony is not increased by the slowing down adaption. If this is not the case, the inter-stream asynchrony is dependent from the delay Δ_i the *Adapt* message suffers from. To assume an upper bound the inter-stream asynchrony is bounded by

   $$\frac{1}{2} \cdot j_M \cdot c. \tag{31}$$

[11] It is supposed here that the asynchrony is nor increased by another $T_{r,M}$ as the slave's media time advances in every case for the $T_{r,M}$ in this stream during the adaption phase.

B. Analysis in Case of Master Switching

Master Switching occurs, whenever a slave stream enters a critical situation, i. e., the buffer of the slave stream is in risk of buffer underflow or buffer overflow. In case of the minimum delay policy the risk of the slave stream will be buffer underflow. According to [13], the risk of buffer overflow must be treated in a different way than by master switching, i. e., it must be treated locally. As master switching is quite complex and can occur under different circumstances, the different cases will be analyzed independently.

1. Master Switching Due to Policy Switching: In case of policy switching (e. g. , from minimum delay to minimum loss policy) the resulting asynchronies may be quite high, as both policies follow contradicting goals. For this reason, we understand both policies as different synchronization schemes. Changing the synchronization policy will result in asynchronies and the decision to change the policy was approved to be reasonable in spite of the resulting asynchronies. For this case, a decrease in quality due to the change of the synchronization policy will be acceptable.

2. Only one Tentative Master Sends an *Adapt* Message: [13] states for this case "in which it adapts [the] release rate R_2 in a way that its buffer delay can be expected to move out of the critical region." For this reason, we parameterize the width of the adaption region in the tentative master. Be a_{tM} a factor for the aggressiveness of the tentative master denoting how far the buffer fill shall be corrected. If the tentative master wants the adaption phase to be as short as possible within the time

$$\frac{1}{2} \cdot a_{tM} \cdot j_{tM}, \tag{32}$$

no PUs have to be played out to sufficiently delay the presentation and fill the buffer. Hereby j_{tM} denotes the jitter, i. e. the buffer width, in the tentative master. For the tentative master, having the role of the master in this adaption phase, the intra-stream asynchrony can be calculated in analogy to equation (20):

$$T_{r,tM} - \frac{T_{r,tM}}{\left\lceil \frac{\frac{1}{2} \cdot a_{tM} \cdot j_{tM}}{T_{r,tM}} \right\rceil + 1}. \tag{33}$$

If the tentative master stream chooses a length L_{tM} being a multiple of the play-out length for one PU T_{tM} as the length of the adaption phase, the asynchrony is reduced to:

$$T_{r,tM} - \frac{T_{r,tM} \cdot L_{tM}}{\left\lceil \frac{\frac{1}{2} \cdot a_{tM} \cdot j_{tM}}{T_{r,tM}} \right\rceil + L_{tM}}. \tag{34}$$

All other streams in this adaption phase are slave streams and equation (27) calculates their asynchrony:

$$T_{r,S} - \frac{T_{r,S} \cdot \left\lceil \frac{l}{T_{r,S}} \right\rceil}{\left\lceil \frac{\frac{1}{2} \cdot a_{tM} \cdot j_{tM}}{T_{r,S}} \right\rceil + \left\lceil \frac{l}{T_{r,S}} \right\rceil}. \tag{35}$$

Finally, the inter-stream asynchrony (cf. Eq. (30)) is specified by

$$\frac{1}{2} \cdot a_{tM} \cdot j_{tM}. \tag{36}$$

3. Two Tentative Masters Send an *Adapt* Message: The ASP ensures that only one tentative master wins (the tentative master with younger time stamps in his message). It is possible that a stream agent receives the older *Adapt* message first and enters the adaption phase. This adaption is intermediate and must be corrected when the slave stream receives the younger *Adapt* message that is valid. Let j_{tM1} and a_{tM1} be the values for the first message received (that is the one that provokes an intermediate adaption phase only) and j_{tM2} and a_{tM2} the values for the other tentative master. Three cases are distinguished:

1. $a_{tM1} \cdot j_{tM1} < a_{tM2} \cdot j_{tM2}$:
   The stream is to be corrected in the right direction, but not fast enough. If the stream has not already entered the adaption phase, asynchrony values are analogous to those given by equations (23), (27), and (30) with values of the second tentative master. In case the stream already entered the adaption phase, these equations still hold, but the phase length is the length of the time interval between the arrival of the *Adapt* message of the second master and the end of its adaption phase. The time the play-out still needs to be delayed determines the correction factor for the adaption phase. As an upper bound $a_{tM2} \cdot j_{tM2}$ might be taken.

2. $a_{tM1} \cdot j_{tM1} = a_{tM2} \cdot j_{tM2}$:
   There are no conflicts between the different adaption phases propagated by the two tentative masters and equation 23, 27, and 30 hold without restriction.

3. $a_{tM1} \cdot j_{tM1} > a_{tM2} \cdot j_{tM2}$:
   Both adaption messages go into the same direction, but the intermediate one corrects the stream too much. If the correction is not yet fulfilled beyond the goal of the second tentative master, the intra-stream asynchrony is bounded by the intra-stream asynchrony of the first tentative master and the inter-stream asynchrony is bounded by the value of the second master. It might also happen that the adaption is already beyond the adaption demanded by the second master. This case is not specifically mentioned in [5] or [13]. We assume here, that the play-out is stopped and resumed, when the goal of the adaption phase of the second master is reached. That is, a synchronization error occurs as the play-out is stopped and the intra-stream asynchrony is bounded by the value for the first adaption message. The maximum additional inter-stream asynchrony is specified by:

$$a_{tM1} \cdot j_{tM1} - a_{tM2} \cdot j_{tM2}. \tag{37}$$

In case the *Adapt* message arrives after the end of the propagated adaption phase, a synchronization error occurs.

This analysis also holds for more than 3 tentative masters, whereas the tentative master that controls the wining adaption phase as well as the fastest of the other tentative masters are used for the analysis.

4. One Tentative Master and the Master Send an *Adapt* Message: If the master initiates an adaption phase to slow down the play-out rate, i. e. to avoid the buffer underflow, the analysis of case three (two tentative masters send an *Adapt* message) is valid. If the master initiates an adaption phase to speed up the play-out rate and to adapt to a new minimum delay, the *Adapt* messages of the master and the tentative master are contradicting. In this case, the maximum inter-stream asynchrony is given by

$$\frac{1}{2} \cdot a_{tM} \cdot j_{tM} + \frac{1}{2} \cdot j_M \cdot c, \tag{38}$$

i. e., the sum of the inter-stream asynchronies in case of an adaption phase in master switching and an *regular* adaption phase. The intra-stream asynchrony, while performing the adaption initiated by the master, is specified by equation (27). If this play-out has to be slowed down, because the stream agent received the winning adaption message of the slave, the intra-stream asynchrony is either bound to

$$T_{r,S} - \frac{T_{r,S} \cdot \left\lceil \frac{l}{T_{r,S}} \right\rceil}{\left\lceil \frac{c \cdot j_M \cdot \frac{1}{2} + \frac{1}{2} \cdot a_{tM} \cdot j_{tM}}{T_{r,S}} \right\rceil + \left\lceil \frac{l}{T_{r,S}} \right\rceil} \tag{39}$$

or 0, in case this is defined as a synchronization error.

6.1.2 Maximum Delay/Minimum Loss Policy

The maximum delay policy tries to guarantee for the minimum data loss by playing out with the maximum delay possible. The data stream with the current minimum delay becomes the master stream. The target area is defined as $UTB = HWM$ and $LTB = HWM - \omega$, where ω denotes the jitter of $d_B(t)$. The analysis of this policy follows the same scheme as the analysis of the minimum delay policy. If a risk of buffer overflow exists in the master a speeding up adaption is initiated. If the buffer fill is lower than the target area, a slowing down adaption is initiated. The master switching is analyzed analogously, whereas the slaves will initiate a speeding up adaption. Results for the analysis in case of no master switching are identical. In case of master switching, cases 1 and 4 lead to the same results.

Case 2: Only one Tentative Master Sends an *Adapt* Message: In case of minimum loss policy we also need to parameterize the width of the adaption region in the tentative master. Let a_{tM} denote this width. If the tentative master wants the adaption phase to be as short as possible, the additional asynchrony in the tentative master is specified by

$$T_{r,tM} - \frac{T_{r,tM}}{\left\lceil \frac{\frac{1}{2} \cdot a_{tM} \cdot j_{tM}}{T_{r,tM}} \right\rceil + 1}, \tag{40}$$

which is derived analogously to equation (20).

If the tentative master chooses length $L_{tM} \cdot T_{r,tM}$ for the adaption phase, its intra-stream asynchrony is bounded by (cf. Eq. (23))

$$T_{r,tM} - \frac{T_{r,tM} \cdot L_{tM}}{\left\lceil \frac{\frac{1}{2} \cdot a_{tM} \cdot j_{tM}}{T_{r,tM}} \right\rceil + L_{tM}}. \tag{41}$$

The intra-stream asynchrony in the slaves, i. e., in all other data streams including the master is specified analogously to equation (27) as

$$T_{r,S} - \frac{T_{r,S} \cdot \left\lceil \frac{l}{T_{r,S}} \right\rceil}{\left\lceil \frac{\frac{1}{2} \cdot j_{tM} \cdot a_{tM}}{T_{r,S}} \right\rceil + \left\lceil \frac{l}{T_{r,S}} \right\rceil}. \tag{42}$$

The inter-stream asynchrony is bounded by

$$\frac{1}{2} \cdot a_{tM} \cdot j_{tM}. \tag{43}$$

Case 3: Two Tentative Masters Send an *Adapt* Message: Also three cases have to be distinguished.

1. $a_{tM1} \cdot j_{tM1} < a_{tM2} \cdot j_{tM2}$:
   The stream is to be corrected in the right direction but not fast enough. If only the adaption phase initiated by the second tentative master is performed, equation (26), (29), and (31) calculate the asynchronies. In case the stream already entered the adaption phase of the first master, these equations still hold, but the length of the adaption phase is the time interval between the correct *Adapt* message was received and the end of its adaption phase. As an upper bound for the time to speed up the presentation $a_{tM2} \cdot j_{tM2}$ can be taken.

2. $a_{tM1} \cdot j_{tM1} = a_{tM2} \cdot j_{tM2}$:
   There are no conflicts between the different adaption phases.

3. $a_{tM1} \cdot j_{tM1} > a_{tM2} \cdot j_{tM2}$:
   Both adaption messages go into the same direction, but the intermediate one corrects the stream too much. If the correction if not yet fulfilled beyond the goal of the second tentative master, the intra-stream asynchrony is bounded by the intra-stream asynchrony of the first tentative master and the inter-stream asynchrony is bounded by the value of the second master. It might also happen that the adaption is already beyond the adaption demanded by the second master. In this case we assume, as stated above, that the play-out is stopped and resumed, when the goal of the adaption phase of the second master is reached. That is, a synchronization error occurs, as the play-out is stopped, and the intra-stream asynchrony is bounded by the value for the first adaption message. The maximum additional inter-stream asynchrony is specified by

$$a_{tM1} \cdot j_{tM1} - a_{tM2} \cdot j_{tM2}. \tag{44}$$

The case that an *Adapt* message arrives after the end of the propagated adaption phase denotes an synchronization error as well.

6.1.3 Clock Offset and Drift

Clocks that have an offset, but do not drift do not result in additional asynchrony. Clocks that drift, where the drift is bounded to ϵ, result in an additional asynchrony of ϵ for the intra-stream synchronization of each stream as well as for the inter-stream synchronization of the synchronization group. Hereby it is assumed that the drift of the clocks is bounded to ϵ. [13] assumes that the clock drift is bounded to ϵ in comparison with global time, therefore, they have an effect of clock drift of an additional asynchrony of $2 \cdot \epsilon$.

6.1.4 Drifting Production and Service Rate

It is possible that the production and the service rate are not equal, e. g., due to unsynchronized clocks. Be $\tau_r = M_r^{-1}(T_r^r)$ and $\tau_s = M_s^{-1}(T_r^s)$ and $\tau_r \neq \tau_s$ in one stream. In this case the receiving stream agent risks buffer underflow (it $\tau_r < \tau_s$) or buffer overflow (if $\tau_r > \tau_s$) and adaption phases are introduced on a *regular* basis.

In this case, we examine the case that an adaption phase is to be triggered due to drifting rates only. Thus, it is assumed that PUs arrive with a constant delay in the receiver and that its buffer fill is in the middle of the buffer area or target buffer area (depending if the stream is the master stream or the slave stream). The number of PUs displayed in the receiver in time τ_s[12] is

$$1 + \frac{\tau_r}{\tau_s - \tau_r} \tag{45}$$

Be x the number of PUs that can be displayed until an adaption phase is triggered. x is determined by

$$(x+1) \cdot \left| \frac{\tau_r}{\tau_s - \tau_r} \right| - \frac{1}{2} \cdot W < 0, \tag{46}$$

where $W = j_M \cdot c$ or $W = a_S \cdot j_S$, respectively, in case the stream is the master or a slave stream. Every $x \cdot \tau_r$ seconds an adaption phase will be triggered.

This time can be used to calculate the probability of an adaption phase for one single stream. Of course, this adaption phase will influence other streams and their requirements to an adaption phase due to drifting rates. The calculation will get the more complex the more different systems with different local time systems are involved and it highly depends on the concrete configuration.

6.2 Synchronization Error Probability

The synchronization error probability is quite difficult to calculate. Of course, all packets, arriving after the maximum total end-to-end delay elapsed, are thrown away and count as synchronization error. Even with a given delay density function of the underlying network, this cannot be calculated easily, as the maximum accepted total end-to-end delay depends on the adaptions that already have been undertaken as well as on the synchronization policy and the stream currently being the master stream.

6.3 Maximum Delay

The maximum delay of each stream as well as of the whole synchronization group is defined as the maximum delay suffered by all streams. But this delay is not constant. If the network delay is changing, the delay is adapted by adapting the play-out rate for a given time interval. For the minimum delay policy it is

$$\Delta_{min} + \frac{1}{2} \cdot (UTB - LTB), \tag{47}$$

whereas Δ_{min} is the minimum delay in the master stream and the master stream is the stream with the maximum minimum delay. For the minimum loss policy the delay is

$$\Delta_{max} - \frac{1}{2} \cdot (UTB - LTB), \tag{48}$$

[12]Note that τ_s is taken here to express the difference in production and service rate. If the rates were equal, the number of PUs displayed in the receiver within a time interval of length τ_s would be 1.

whereas Δ_{max} is the maximum delay of the master stream and the master stream is the stream with the minimum maximum delay.

For the delay during adaption phases, no equation can be specified, as this depends highly on the stream initiating the adaption phase and the progress state of the adaption phase.

6.4 Buffer Requirements

The buffer requirement in each stream is dependent on the size of the play-out buffer in each stream. It is slightly larger than the buffer region between HWM and LWM, as there is a critical buffer region below LWM as well as above HWM. Due to the fact the algorithm adapts to the underlying network delay, the buffer delay can be reduced compared to rigid algorithms, if the network delay changes slowly, without risk of extreme quality reduction due to numerous adaption phases in short time.

7 Results of the Analysis of Two Example Algorithms

7.1 Analyzing the Concord Algorithm

The Concord algorithm is a rigid synchronization algorithm. The synchronization is achieved totally distributed, i. e., there is no communication between the single data streams. The Concord algorithm assumes a given knowledge on the network infrastructure as well as periodic data.

The parameters used in the analysis are the following:

Δ_0	be the network delay of the first packet
λ	be the error in estimating the network delay of the first packet
ϵ	be the bound for the clock asynchrony
Δ_{min}	be the minimum network delay
Δ_{opt}	be the maximum accepted delay (respecting the delay distribution)
T_r	play-out period in the receiver (in the local time system)
τ_r	play-out period in the receiver (in the global time system)
τ_s	play-out period in the sender (in the global time system)

7.1.1 Results for One Single Stream

The algorithm has the following characteristics:

basic algorithm	
asynchrony	0
delay	Δ_{opt} is depending on the delay distribution of the network
synchronization error	specified packet loss rate
buffer requirements	$\left\lceil \frac{\Delta_{opt}-\Delta_{min}}{T_r} \right\rceil$ packets

unknown network delay	
$\lambda \leq T_r$	
asynchrony	0
delay	$\Delta_{opt} + \lambda$
synchronization error	packet loss rate
buffer requirements	$\lceil \frac{\Delta_{opt} - \Delta_{min}}{T_r} \rceil$ packets
$\lambda > T_r$	
asynchrony	$\lfloor \frac{\lambda}{T_r} \rfloor \cdot T_r$
delay	first $\Delta_{opt} + \lambda$ (then reducing to $\Delta_{opt} + \lambda - \lfloor \frac{\lambda}{T_r} \rfloor \cdot T_r$)
synchronization error	in total $\lfloor \frac{\lambda}{T_r} \rfloor$ packets being thrown away additionally
buffer requirements	$\lceil \frac{\Delta_{opt} - \Delta_{min}}{T_r} \rceil$ packets

clock offset and drift	
asynchrony	ϵ
delay	in $[\Delta_{opt} - \epsilon, \Delta_{opt} + \epsilon]$ (thus, maximum bound: $\Delta_{opt} + \epsilon$)
synchronization error	no additional
buffer requirements	$\lceil \frac{\Delta_{opt} + \epsilon - \Delta_{min}}{T_r} \rceil$

different production and service rate	
asynchrony	$2 \cdot \tau_r - \tau_s$ or τ_s respectively
delay	$\tau_r > \tau_s$: $\Delta_{opt} + \tau_r$ Δ_{opt} else
synchronization error	$\frac{1}{\frac{\tau_r}{\lvert \tau_r - \tau_s \rvert}} = \frac{\lvert \tau_r - \tau_s \rvert}{\tau_r}$-th of packets are additionally thrown away or duplicated
buffer requirements	$\lceil \frac{\Delta_{opt} - \Delta_{min}}{T_r} \rceil$

In an environment that has, e. g., unknown network delay and different production and service rates, the values for the QoSy parameters have to be combined to assess the algorithm for the specific environment.

7.1.2 Results for the Synchronization Group

The parameters used in the table are the following:

Δ_g	be the end to end delay for the synchronization group
ϵ	be the bound for the clock asynchrony
$\Delta_{i,min}$	be the minimum network delay of stream i
$\tau_{r,i}$	be the play-out period in the receiver for stream i
$p_i(err)$	be the packet error probability in stream i

asynchrony	results in the maximum asynchrony of the streams $+\epsilon$
delay	Δ_g results in the maximum delay of the streams
synchronization error	$1 - \prod_i (1 - p_i(err))$, i are the streams
buffer requirements	$\sum_i \lceil \frac{\Delta_g - \Delta_{i,min}}{T_{i,r}} \rceil$

7.2 Analyzing the Adaptive Synchronization Protocol

The Adaptive Synchronization Protocol implements an adaptive synchronization mechanism with centralized control. Data is required to be periodic. ASP follows the master/slave synchronization approach. It adapts the total end-to-end delay of the master stream, and of the slave streams, to the underlying network delay. In doing so, there are multiple parameters that influence the behavior of ASP: the buffer smoothing function, the width of the buffers in all streams, the width of the target buffer area in the master stream, the aggressiveness of the adaption process in the master and the tentative masters, especially the length of the adaption phases, and the synchronization policy. For this reason, ASP describes a class of protocols and, therefore, the following results are parameterized.

The following parameters are used in the analysis:

Δ_{min}	be the minimum network delay in the master stream
Δ_{max}	be the maximum network delay in the master stream
c	be the width of the target buffer area (1 in case of a tentative master)
j_M	be the by the master stream accepted jitter
j_S	be the by the slave stream accepted jitter
L_M	be the length of the adaption phase as defined by the master
l	describing the time resting in the slave stream to carry out an adaption phase
$T_{r,M}$	be the play-out period in the master stream
$T_{r,S}$	be the play-out period in the slave stream
ϵ	be the bound for the clock asynchrony

7.2.1 Results for One Single Stream

The results of the ASP are summarized in the following tables.

in the master/tentative master	
no adaption	
asynchrony	0
delay	$\Delta_{min} + \frac{1}{2} \cdot c \cdot j_M$ (minimum delay policy) or $\Delta_{max} - \frac{1}{2} \cdot c \cdot j_M$ (minimum loss policy)
slowing down adaption	
asynchrony	$\frac{\frac{1}{2} \cdot c \cdot j_M}{L_M}$
delay	increasing
speeding up adaption	
asynchrony	$T_{r,M} - \frac{L_M \cdot T_{r,M}}{\left\lceil \frac{\frac{1}{2} \cdot c \cdot j_M}{T_{r,M}} \right\rceil + L_M}$
delay	decreasing
for all	
synchronization error	depending on the underlying network and current delay not increased by the algorithm
buffer requirement	minimal $\left\lceil \frac{j_M}{T_{r,M}} \right\rceil + 2$ PUs as there need to be place in the critical buffer region outside the target buffer

in the slave	
no adaption	
asynchrony	0
delay	$\Delta_{min} + \frac{1}{2} \cdot c \cdot j_M$ (minimum delay policy) or $\Delta_{max} - \frac{1}{2} \cdot c \cdot j_M$ (minimum loss policy)
slowing down adaption	
asynchrony	$\frac{\frac{1}{2} \cdot c \cdot j_M}{\left\lceil \frac{l}{T_{r,S}} \right\rceil}$
delay	increasing
speeding up adaption	
asynchrony	$T_{r,S} - \frac{T_{r,S} \cdot \left\lceil \frac{l}{T_{r,S}} \right\rceil}{\left\lceil \frac{\frac{1}{2} \cdot c \cdot j_M}{T_{r,S}} \right\rceil + \left\lceil \frac{l}{T_{r,S}} \right\rceil}$
delay	decreasing
for all	
synchronization error	depending on the underlying network and current delay not increased by the algorithm
buffer requirement	minimal $\left\lceil \frac{j_S}{T_{r,S}} \right\rceil + 2$ PUs as there need to be place in the critical buffer region outside the target buffer

clock offset and drift	
asynchrony	$+2 \cdot \epsilon$
delay	no additional
synchronization error	potentially in case the drift leads to additional adaption phases
buffer requirement	no additional

drifting production and service rate	
asynchrony	no additional
delay	no additional
synchronization error	additional, as this leads to additional adaption phases
buffer requirement	no additional

7.2.2 Results for the Synchronization Group

no adaption	
inter-stream asynchrony	0 or ϵ in case of bounded clocks
delay	$\Delta_{min} + \frac{1}{2} \cdot c \cdot j_M$ or $\Delta_{max} - \frac{1}{2} \cdot c \cdot j_M$ respectively
slowing down adaption	
inter-stream asynchrony	0 if adaption phase is delayed $\frac{1}{2} \cdot c \cdot j_M$
delay	increasing
speeding up adaption	
inter-stream asynchrony	0 if adaption phase is delayed $\frac{1}{2} \cdot c \cdot j_M$
delay	decreasing
for all	
synchronization error	depends on the buffers in the streams, the network delays, the fact, if the adaption phase is delayed until all slaves received the *Adapt* message, and how often adaption phases have to be performed
buffer requirements	the sum of the buffer requirements in the single streams

8 Conclusions and Future Work

In this report *Quality of Synchronization (QoSy)* is defined as a concept describing the synchronization quality independent from the environment and the application. QoSy comprises the four parameters asynchrony, synchronization delays, buffer requirements, and synchronization error probabilities. The time model as defined in this report allows for the definition of the asynchrony in case of intra-stream and in case of inter-stream synchronization. An analysis model for distributed synchronization is introduced. This model allows to use QoSy parameters to analyze synchronization algorithms and assess their quality. The analysis has been carried out for one rigid and one adaptive synchronization algorithm and is described in detail in this report. The results obtained proved that QoSy is suitable to study characteristics of synchronization algorithms in detail. The concept is general enough to be used for different synchronization approaches and allows to analyze them independently from the environment.

During the analysis not only quantitative results as presented in Chapter 7 were obtained, but also characteristics of application, user requirements, end-system, and communication system influencing synchronization methods have been determined. Figure 3 indicates these qualitative results. Influencing parameters must be integrated into a calculation of QoSy to obtain the quality criteria for a given synchronization method in a given environment and for a given application.

Within a given environment characteristics of the Concord algorithm can be directly derived. In case of the ASP protocol this is not so easy, as there are many interdependent and changing parameters influencing the behavior of the algorithm. The asynchrony calculated for the adaption phases helps to dimension the buffer size, the target area size, and the length of the adaption phase such to not exceed a user specified value for the asynchrony.

The analysis has been carried out for unicast applications only including live data.

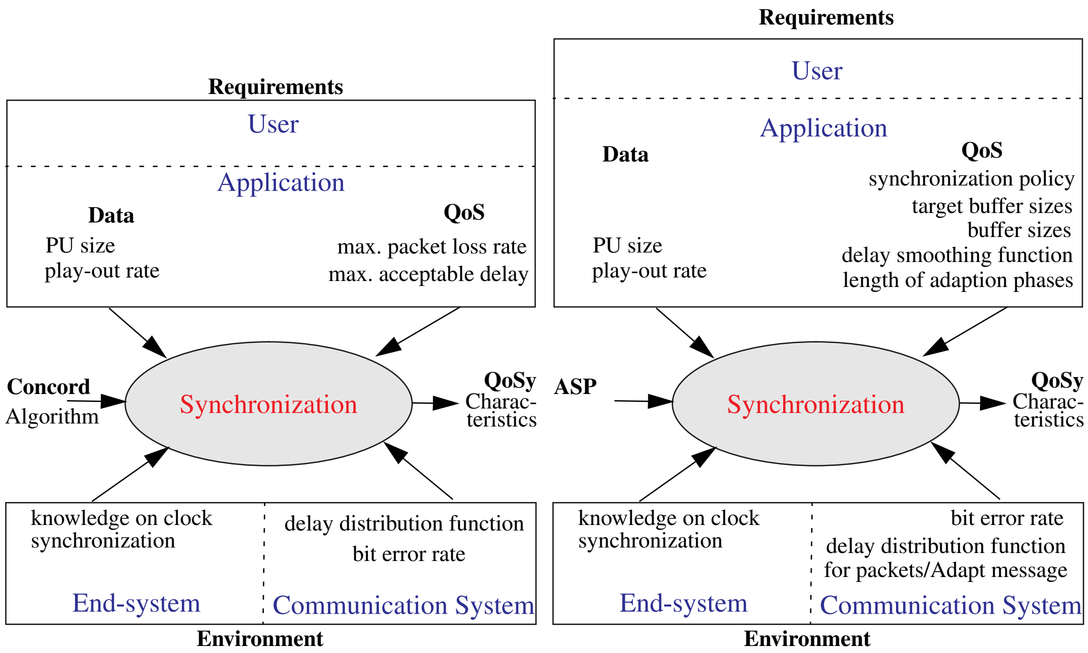

Figure 3: The Analysis Model Applied to both Example Algorithms

The fact that data may be retrieved in advance and can be stored in the receiver has not yet been taken into account. In a next step of our work we want to measure the quality criteria in a simulation to substantiate our results. QoSy criteria will be studied in multi-party applications especially in video-conferencing including a quite critical delay.

References

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

[2] 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.

[3] C. Class. Synchronization Issues in Distributed Applications: Definitions, Problems, and Quality of Synchronization. Technical Report 31, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland, December 1997. Available at URL: ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report31.ps.

[4] 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.

[5] T. Helbig. *Kommunikation und Synchronisation multimedialer Datenströme in verteilten Systemen*. PhD thesis, Universität Stuttgart, Germany, June 1996.

[6] International Telecommunications Union Telecommunications Standardization Sector, Recommendation X.902. *Information Technology - Open Distributed Processing - Reference Model*, November 1995.

[7] 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.

[8] K. Nahrstedt and R. Steinmetz. Resource Management in Networked Multimedia Systems. *IEEE Computer*, 29(5):52–63, May 1995.

[9] 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, SPIE, February 1995.

[10] 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, SPIE, February 1995.

[11] 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.

[12] 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.

[13] K. Rothermel and T. Helbig. An adaptive protocol for synchronizing media streams. *Multimedia Systems*, 5(5):324–336, September 1997.

[14] 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.

[15] 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.

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

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

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

[19] 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.

[20] M. Zitterbart, B. Stiller, and A. Tantawy. A Model for Flexible High-Performance Communication Subsystems. *IEEE Journal on Selected Areas in Communications*, 11(4):507–518, May 1993.