Evaluation of how clock synchronisation protocols affects inter-sender synchronisation of live continuous multimedia

Emil Nilsson
Abstract

There exists several inter-sender synchronisation algorithms today, some of these requires globally synchronised clocks. Exactly how the clocks are going to be synchronised are generally not mentioned. This thesis aims to investigate how the choice of clock synchronising protocol affects these kinds of synchronisation algorithms. Tests have been conducted with NTP and PTP on a rudimentary inter-sender synchronisation algorithm. Results shows that there is little difference between protocols and the choice between them should be done on what constraint each protocol introduces. However if performance is extremely important there is some benefits to choosing a better clock synchronisation protocol.
Acknowledgements

First and foremost I would like to thank Björn at Intinor, who has been my external supervisor and an invaluable asset. Without you this thesis would probably not have been completed. Furthermore I would like to thank the work colleagues at Intinor, my internal supervisor Mikael Rännar, Ludvig and Alexander who has been proof reading the thesis and all my friends and family.
6 Discussion

6.1 Results

6.1.1 Implementation

6.1.2 Clock synchronisation protocols

6.2 Practical implications of clock synchronising protocols

6.2.1 Hardware

6.2.2 Placement

6.2.3 Other constraints

6.3 GPS failure

7 Conclusion and future work

7.1 Future work

References
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-frame</td>
<td>Bidirectional picture</td>
</tr>
<tr>
<td>BMC</td>
<td>Best Master Clock algorithm</td>
</tr>
<tr>
<td>DTS</td>
<td>Decoding Timestamp</td>
</tr>
<tr>
<td>ES</td>
<td>Elementary Stream</td>
</tr>
<tr>
<td>FPS</td>
<td>Frames Per Second</td>
</tr>
<tr>
<td>GOP</td>
<td>Group Of Pictures</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>I-frame</td>
<td>Intra-coded picture</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter Process Communication</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MPEG-PS</td>
<td>Moving Picture Experts Group - Program Stream</td>
</tr>
<tr>
<td>MPEG-TS</td>
<td>Moving Picture Experts Group - Transport Stream</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>P-frame</td>
<td>Predicted picture</td>
</tr>
<tr>
<td>PES</td>
<td>Packetized Elementary Stream</td>
</tr>
<tr>
<td>PID</td>
<td>Packet Identifier</td>
</tr>
<tr>
<td>PPS</td>
<td>Pulse Per Second</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision Time Protocol</td>
</tr>
<tr>
<td>PTS</td>
<td>Presentation Timestamp</td>
</tr>
<tr>
<td>SCF</td>
<td>System Clock Frequency</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>UTC</td>
<td>Coordinated Universal Time</td>
</tr>
<tr>
<td>QoE</td>
<td>Quality of Experience</td>
</tr>
</tbody>
</table>

**Table 1:** List of abbreviations used in this thesis.
1 Introduction

This section introduces this thesis, its scope and its goal. Also an outline for the thesis is provided.

1.1 Background

Enormous advancements on IP networks, as well as the widespread deployment of the internet has fuelled the growth of multimedia distribution over IP-networks for the last two decades [7]. The same advancements has also made it feasible to perform multimedia contribution over IP networks. The difference of the two being that contribution is delivering multimedia from point A to point B, meanwhile distribution is about delivering multimedia to multiple end users. Generally multimedia is contributed from the recording location to a studio where it is mixed and later distributed to the users.

First of all, a distinction between continuous and discrete multimedia has to be done. The former is characterised by sequences of time correlated media packets, generated by different sensors over time (e.g. video or audio). The latter regards static media data (e.g. a single image) or standalone media events (e.g. text animation).

Preserving temporal relations of continuous multimedia during contribution is important. This is because errors introduced during contribution will be propagated to distribution and thus to the end users. Introducing a noticeable error may cause multimedia to be perceived as artificial, strange or even annoying[13]. Thus severely degrading the Quality of Experience (QoE) for the end user. Preserving temporal relations of multimedia is known as multimedia synchronisation.

To be able to perform multimedia synchronisation, it is essential to know what destroys temporal relations of multimedia. Generally the inherently unreliable IP network is a big cause. It introduces factors such as: network delays, network jitter and network skew. Furthermore, IP networks are used to transmit data between at least two computers. Two different computers have two different physical clocks that may have a skew as well as they may drift differently in time. If only one computer was involved, this would not pose a problem, since all components would have access to the same clock. However IP networks are not the only cause of asynchrony, end-system jitter and rate drift also causes asynchrony in multimedia.

It is possible to divide multimedia synchronisation into the following four layers: Intra-stream synchronisation, Intra-media synchronisation, Intra-bundle synchronisation and Intra-session synchronisation [5].

Intra-stream synchronisation is the lowest layer. It arranges a single media stream’s Media
Frames (MF’s) in correct order. Examples of a MF is a video frame within a video stream, or an audio sample within an audio stream. A synchronisation failure in this layer may cause temporal media distortion (e.g. audio crackle or image jerkiness).

Intra-media synchronisation handles synchronisation of multiple streams originating from devices with the same media modality. Media modality refers to what type of media the device captures, it could for instance be video or audio. Failure to synchronise at this layer might lead to violations of spatial correlations of media (e.g. visual mismatch between multiview images).

Intra-bundle synchronisation (or inter-stream synchronisation) handles synchronisation of multiple streams originating from devices with different media modality. A frequently studied example of this is the lip synchronisation problem, which consists of synchronising an audio stream and a video stream. A failure here might cause audio to be played after the corresponding video.

Intra-session synchronisation represents both inter-sender synchronisation and inter-receiver synchronisation. Although only one need to be used for it to be classified as intra-session synchronisation. Inter-receiver synchronisation (also named group synchronisation and inter-destination synchronisation) refers to synchronisation of the same media at multiple receivers. Failure to synchronise this may lead to unfairness when multiple people at different receiver sites gets a timing privilege to conduct an activity. Inter-sender synchronisation conducts synchronisation from multiple senders at the same receiver. A synchronisation failure here may lead to the receiver being confused, if the senders is conducting a highly collaborative activity.

1.2 Goal

There exists several different techniques and algorithms for continuous multimedia synchronisation [6]. Some of these algorithms requires globally synchronised clocks between all senders. Exactly how the clocks are going to be synchronised are, however, not mentioned. Thus the choice of clock synchronisation protocol, if such is required, is neglected. Neglecting this choice may affect the performance of the algorithm. This thesis aims to investigate how the choice of clock synchronisation protocol affects the inter-sender synchronisation algorithm.

1.3 Scope

To investigate the impact of the choice of clock synchronisation algorithm, a rudimentary inter-sender synchronisation algorithm designed for live continuous media has been implemented. Thus only inter-sender synchronisation will be considered in this thesis. Intra-stream and intra-bundle synchronisation is, of course, performed in the implementation, but not evaluated in the results. Inter-receiver synchronisation will not be considered at all.

Tested clock synchronising protocols has been limited to two, to reduce the scope. Only

---

1 It does not exist a widespread name for this. Other names for a MF may be Media Unit (MU), Media Data Units (MDU), Access Unit (AU) or Logical Data Unit (LDU) depending on the author and/or the application.
Network Time Protocol (NTP) and Precision Time Protocol (PTP) have been tested. This choice was motivated by the difference in precision between protocols. PTP provides high precision, meanwhile NTP focuses on accuracy and precision comes in second hand. Global Positioning System (GPS) was also intended to be tested, but hardware problems made it not possible. However even though GPS was not tested, it is included in the discussion.

This thesis has been developed in cooperation with Intinor. Intinors hardware and software has been the foundation the implementation was build on. Having this foundation introduced a limitation on the implementation. This limitation being that video and audio streams have to be delivered with Moving Picture Expert Group - Transport Stream (MPEG-TS). Thus reducing the options on how information derived from the synchronised clocks (i.e. timestamps) may be delivered.

### 1.4 Outline

This thesis starts of with a theory chapter that introduces the theory about synchronisation as well as protocols used in this thesis. After that a chapter that explains how the implementation works is presented. Then the methods used for the experiments are described, and after that the results are presented. At last a discussion about the results and future work takes place. Abbreviations used in the thesis can be looked up before this introduction, in Table 1.
2 Theory

This chapter introduces protocols used in this thesis and provides general theory on synchronisation.

2.1 Clock synchronisation

Sender and receiver are located at different physical devices, and thus they have separate physical clocks. Due to cost restrictions these clocks are not perfect. These imperfections as well a temperature differences makes them subject to clock drift as well as clock skew. Even if the clock skew is perceived as small at first, it will grow big over time. To combat the existing clock drift, clocks need to be continuously synchronised by some clock synchronisation protocol.

Before going through the protocols, the difference between an **accurate** and a **precise** clock synchronisation protocol should be explained. An accurate clock synchronisation protocol ensures that the clocks are close to real time. A precise clock synchronisation protocol, however, ensures that all clocks that is being synchronised are close together in time, regardless of the real time. Thus accuracy does not always imply precision and vice versa. However accuracy can be used to get a fairly good precision. This is because all clocks are close to real time, and thus they are fairly close to each other.

In this thesis the important characteristic of a clock synchronisation protocol is precision. All senders needs to be closely synchronised to the same time. This is to be able to compare timestamps between senders, and calculate a time difference between two frames originating from different senders. Thus it does not matter if the time is close to the real time or not.

2.1.1 Network Time Protocol

Since its introduction in the 1980’s, NTP has allowed millions of computers to synchronise their clocks. It is today one of the oldest internet protocols still in use. Currently NTP is at version 4, defined in RFC5905 [10].

NTP divides all participating computers in a hierarchy, where each level is called a **stratum**. In total there exists 16 stratum, where a lower number indicates a stratum higher up in the hierarchy. Every participating computer synchronises their clock towards computers at a stratum exactly one higher or at the same stratum. However this does not apply to stratum 0, since it is at the top of the hierarchy. At stratum 0 **reference clocks** are located. A reference clock is a clock that is very accurate in keeping time, it could be an atomic clock, a GPS, or a radio clock. Computers at stratum 1 is called primary time servers, because they synchronise directly with a reference clock. Due to the structure of NTP, time will be
less accurate when venturing down in the hierarchy.

To synchronise clocks a client calculates time offset ($\theta$) and round trip delay ($\delta$) towards a NTP server. The equations used for these calculations can be seen in Equations 2.1 and 2.2. To be able to perform these calculations, the client sends a request to the server. In the packet sent, the client stores a transmission timestamp ($T_1$). When the server receives a request, it will add a receive timestamp ($T_2$) and a transmit timestamp ($T_3$) to the packet before sending back the package. Upon receiving the reply, the client will save the receive timestamp ($T_4$). This packet exchange is illustrated in Figure 1. When the client has received the reply it can calculate the time offset and round trip delay to the server.

$$\theta = \frac{(T_2 - T_1) + (T_3 - T_4)}{2} \quad (2.1)$$

$$\delta = (T_4 - T_1) - (T_3 - T_2) \quad (2.2)$$

The calculated values are passed through filters and subjected to statistical analysis. Before a client trusts a server, several successful exchanges needs to have taken place. When a client trusts a server it starts to adjust its clock gradually towards the servers clock.

Generally a client synchronises time with more than one server. In case of server disagreements, the largest set of agreeing servers is used together to create the reference time. All disagreeing servers are declared as invalid and thus not used for synchronisation.

A problem with NTP is that it assumes that the round trip delay is symmetrical (which is possible to observe in Equation 2.1). This means that the travelling time for a packet is the same for both ways to/from the server. When asymmetrical delays are present NTP performs poorly. For instance if the asymmetric delays are very bad the errors can be as big as 100 milliseconds. Compared to the expected accuracy of 1 millisecond that NTP provides.[9]

### 2.1.2 Global Positioning System

GPS is mostly known for providing location information to its users, however it can also be used to accurately synchronise computer clocks. GPS is the American implementation of a Global Navigation Satellite System (GNSS). Other superpowers has their own positioning system also, although they are less known to the public. Europe has Galileo, Russia
has GLONASS and China has BeiDou. These versions can also be used to synchronise computer clocks.

The GPS system consists of satellites that orbits the earth. To be able to use the system, a GPS receiver is needed. If the receiver is in line of sight of a GPS satellite, it is able to capture a signal that the satellite broadcasts. This signal consists of time and current position of the satellite. To be able to triangulate its position, the receiver needs to be in line of sight of at least 4 satellites at any given time. Since the satellites contains a very stable atomic clock, the time is very accurate and may be used to synchronise the host of the GPS receivers clock. If everything works out as planned, it is possible to synchronise the hosts clock to within 100 nanoseconds of real time [11].

2.1.3 Precision Time Protocol

PTP is defined in IEEE 1588-2008[2] and is currently at version 2. As might be deduced by the name, PTP is a protocol focusing on precision rather than accuracy. Although accuracy can be attained, external help is needed for that specific purpose. PTP is intended to use on a Local Area Network (LAN) with Ethernet connections. It can achieve a precision of 1 microsecond. However if hardware timestamping is used, the precision can go as low as 100 nanoseconds. It is specifically designed to fill a niche, which neither GPS or NTP can fulfil. This nice being local systems that needs higher precision than NTP, meanwhile having a cost restriction and thus not able to afford a GPS unit for each node.

PTP uses a master slave architecture to synchronise the clocks within the network. The master is considered to have the correct time and all slaves will be synchronised towards it. If the masters clock is synchronised with high accuracy with for instance a GPS, this accurate time will be propagated within the network and thus accuracy might be achieved for all nodes. To be able to solve the disappearance of a master, PTP has implemented a Best Master Clock algorithm (BMC) that will choose a new master without any human interaction. This gives PTP much robustness and avoids a single point of failure. The disappearance of a master might be problematic if it was synchronised to reference time and the new master is not.

Clock synchronisation is done in a similar manner as NTP. Opposite to NTP, PTP preferably runs in multicast mode, which makes the master the initiator of the synchronisation process. To start synchronising, the master multicasts a SYNC-message over the network. If hardware timestamping is supported, this message will contain a timestamp when the message was sent. However, if hardware timestamping is not supported a FOLLOW-UP-message with the sending timestamp of the SYNC-message also has to be sent. Each slave receiving these massages are measuring the reception time of the messages. With this information, the slave can calculate the offset towards the master clock. What has been done now is phase 1 in the synchronisation process, and the clocks would be synchronised if there was no delay in the network. Phase 1 is done about once every two seconds.

Phase 2 is run more rarely than phase 1, about every 60 seconds. This is to reduce the load on the network. This phase tries to compensate for the delay in the network. It starts by the slave sending a DELAY-REQUEST-message to the master, and saving the transmission time. When the master has received this message, it responds with a DELAY-RESPONSE-message containing both the receive time of the DELAY-REQUEST-message as well as the sending time of the DELAY-RESPONSE-message. The slave can now calculate the network delay
with the same equation as in NTP, Equation 2.2. Because of this calculation, PTP is also vulnerable to asymmetric network delays. However asymmetric delays are less prominent on LANs due to shorter traveling distances and less competing packages. All messages sent can be viewed in Figure 2.

2.2 MPEG-TS

MPEG-TS is an abbreviation for Moving Picture Experts Group Transport Stream, and is a standard digital container format for storage and transmission of audio and video data. It is intended for transmission of MPEG-1, MPEG-2 and MPEG-4 compressed video/audio data. Compared to its sibling, MPEG-PS (MPEG Program Stream), it is designed for unreliable transmission such as IP networks provide. To combat these unreliable factors, MPEG-TS transmits synchronising information as well as error correction information. However this synchronising information does not have any connection to real time, and thus can only be used to synchronise multiple MPEG-TS streams originating from the same sender.

Packetizing in MPEG-TS is done in two steps. First an Elementary Stream (ES) (e.g. an encoded video stream) is packetized into an Packetized Elementary Stream (PES). After that the PES is packetized into 188 byte big MPEG-TS packets. When this is done, it is time to deliver the MPEG-TS packets to the decoder. This delivery is not a part of MPEG-TS, but common delivery protocols are User Datagram Protocol (UDP) and Transmission Control Protocol (TCP).

2.2.1 Multiplexing

MPEG-TS supports multiplexing of multiple streams. Which means it is possible to send multiple streams through the same MPEG-TS. To be able to identify which stream each MPEG-TS packet belongs to, a 13 bit Packet Identifier (PID) is sent within the MPEG-TS header. A common case when multiplexing is needed is when transmitting a movie through MPEG-TS. Both video and audio are separate streams and if they are to be sent to the same receiver, they may as well be sent in the same MPEG-TS. It is also possible to send multiple TV channels within the same MPEG-TS.

**Figure 2:** Messages sent within the PTP protocol.
2.2.2 Decoding and presentation timings

A PES packet generally consists of a single video frame, or multiple audio samples. But it could also contain part of a video frame, if it is split up in multiple PES packets. Which might be done if interlacing is used. In the header a Decoding Timestamp (DTS) and Presentation Timestamp (PTS) is sent. To understand what DTS and PTS is and why they are needed, some general knowledge of video compression is needed. Generally there exists three different type of frames in a video compression algorithm:

- **I-frame.** I-frame stands for intra-coded picture. This is a complete image, which could be compared to a regular JPEG picture.

- **P-frame.** A P-frame is also known as a predicted picture and is a frame that only holds changes towards the previous anchor frame. Imagine a moving object on a stationary background. Only changes of the object needs to be conveyed, since the stationary information already exists. This saves bandwidth compared to an I-frame. Both I- and P-frames are considered anchor frames.

- **B-frame.** B-frame stands for bidirectional predicted frame. It is similar to the P-frame since it contains changes to the previous anchor frame. However it also contains changes to the succeeding anchor frame, thus saving even more bandwidth.

The frames are often grouped into a Group of Pictures (GOP), which specifies a way of arranging the different frames. A GOP is often represented by two numbers, M and N. M is the distance (in number of frames) between two anchor frames and N is the distance between two I-frames. So, for example, an M = 3 and N = 9 produces the following frame sequence: IBBPBBPBBI. This sequence and how the frames reference each other can be observed in Figure 3.

This relationship between frames introduces a problem to transport streams, which is that frames cannot be delivered to the video decoder in the presentation order. The problem arises because to decode a B-frame, the next anchor frame needs to be decoded. To solve
this problem, the variables DTS and PTS are used. DTS represents when a frame should be sent to the decoder and PTS represents when a decoded frame should be presented. Thus an anchor frame will have an earlier DTS than a preceding B-frame, even though the anchor frames PTS is after the B-frames PTS. Note that the absence of I-, P- and B-frames in audio will always make DTS equal to PTS.

2.2.3 Internal clock

The encoder has an internal 27 MHz clock that both PTS and DTS refers to. This clock is named System Clock Frequency (SCF) and has no connection to real time. Thus, for the decoder to be able to understand a streams DTS and PTS, the clock needs to be reconstructed at the decoder. How the reconstruction is done is dependent on which of MPEG-1, MPEG-2 and MPEG-4 is transported within MPEG-TS. However all methods are similar and only describing MPEG-2 will suffice as an example of how it is done. All methods can be found in [14].

MPEG-2 clock reconstruction

MPEG-TS may transport multiple ES, every one of these has a different timebase. Thus each ES has to convey its own clock information. The clock information consists of a variable named Program Clock Reference (PCR). PCR(i) is defined as “the time \( t(i) \) when the byte that contains the last bit of the PCR fields arrives at the decoder.” A PCR value has to be conveyed in the MPEG-TS at least every 0.1 seconds. To construct the PCR, two fields are combined: \( PCR\_{\text{base}} \) and \( PCR\_{\text{ext}} \). Exactly how the PCR variable is constructed can be seen in Equations 2.3 - 2.7. This calculation also includes the cases when \( PCR\_{\text{base}} \) and \( PCR\_{\text{ext}} \) is not sent and interpolation is needed.

\[
PCR(i) = PCR\_{\text{base}}(i) \times 300 + PCR\_{\text{ext}}(i) \tag{2.3}
\]

Where:

\[
PCR\_{\text{base}}(i) = \left( \frac{SCF \times t(i)}{300} \right) \mod 2^{33} \tag{2.4}
\]

\[
PCR\_{\text{ext}}(i) = \left( \frac{SCF \times t(i)}{1} \right) \mod 300 \tag{2.5}
\]

Consider \( i, i', i'' \) as indices to bytes in MPEG-TS where \( i'' \leq i \leq i' \). Equation 2.6 is applied to calculate at what time any byte \( i \) within MPEG-TS arrives at the decoder \( t(i) \).

\[
t(i) = \frac{PCR(i')}{SCF} + \frac{i - i''}{TR(i)} \tag{2.6}
\]

Where TR is the Transport Rate. TR(i) is the TR for any byte \( i \) between bytes \( i'' \) and \( i' \). TR(i) can be calculated with Equation 2.7

\[
TR(i) = \frac{((i' - i'') \times SCF)}{PCR(i') - PCR(i'')} \tag{2.7}
\]
DTS and PTS can later be related to this time. First they need to be converted from the format sent in the header. It is done by Equations 2.8 and 2.9. Note that the division by 300 makes PTS and DTS compliant with a 90 KHz clock, instead of the 27 MHz. This is because the resolution of a 27 MHz clock is not needed for this synchronisation. The greater resolution is more suitable for synchronising PCR with a remote encoder.

\[
PTS(k) = \left( \frac{SCF \times t_p(k)}{300} \right) \% 2^{33}
\]  

\[
DTS(j) = \left( \frac{SCF \times t_d(j)}{300} \right) \% 2^{33}
\]

In the Equation 2.8 \( t_p(k) \) is defined as “presentation time, measured in seconds, in the decoder, of the \( k \)th presentation unit in ES\(_n\)”. Likewise, in Equation 2.9 \( t_d(j) \) is defined as “the decoding time, measured in seconds, in the decoder, of the \( j \)th access unit in ES\(_n\)”.

### 2.3 Perception of asynchrony

When evaluating multimedia synchronisation, it is essential to know at what thresholds humans start to notice asynchrony. Of course it is very individual, for instance an individual that has a lot of experience in working with audio and video will notice asynchrony at lower thresholds. Asynchrony perception also depends on which media modality that is being presented. For example haptic-to-video and haptic-to-audio have different thresholds. Haptic-to-video has a threshold at \([-87, 125]\) ms meanwhile haptic-to-audio has a threshold of \([-92, 110]\) ms [12].

In this thesis, the use case is that two or more audio/video-streams are being synchronised. Which means that audio-audio, audio-video and video-video synchronisation will take place. The most well studied of these is audio-video synchronisation, also called lip synchronisation. It has been concluded that the thresholds for audio-to-video are \([-80, 80]\) ms [13]. Another interesting finding is that a skew is more easily detected when video was before audio (i.e. negative skew) than the opposite. This may be explained by the fact that humans are more used to see a thing happening before hearing it, because light travels faster than sound.
3 Implementation

MPEG-TS only performs inter-bundle synchronisation, and thus an extension needs to be implemented to facilitate inter-sender synchronisation. This chapter describes how it was implemented.

First of all, a method to insert timestamps of Coordinated Universal Time (UTC) into MPEG-TS was devised. UTC is the primary time standard by which the world regulates its clocks. Furthermore methods utilising this information at the decoder to perform inter-sender synchronisation was also constructed. How the system looks at a coarse level and what modules exists can be viewed in Figure 4.

3.1 Sending timestamps

As said before: MPEG-TS needed to be modified someway, to include UTC timestamps. A big problem that arises when modifying a standardised protocol is that it will likely break all decoding that other, already existing, decoders have implemented. This should, preferably, not happen. Thus not breaking the protocol was prioritised highly, and a solution that manages to not break decoding by other decoders as well as propagating a timestamp to the decoder was invented.

The solution attaches the timestamp in the 128 bit long private data field of the PES header. This field is meant to send user specified information within MPEG-TS, which makes it a good candidate for conveying a timestamp. Imaginably the field is perfect for inserting a 128 bit timestamp. However it is not, since the sequence 0x000001 may never occur within a MPEG-TS. Thus, marker bits have to be inserted every second byte and this shortens down the actual data capacity of the field. To reduce the size of the timestamp, the number of bits are reduced to 64. This reduction introduces the 2038 problem, which is that the timestamp only can represent time to 2038.

The 2038 problem is not a big issue with this implementation, because it does not matter for the encoders that they have the real UTC time. What matters is that they have the same time between each other. Using UTC is just an "easy" way of achieving the same time at different encoders. When the flip occurs at 2038 to 1970 (where the time definition starts), it is only a problem for as long as only one has flipped. When both have flipped, they have the same time again. This problematic period of time is probably very low, since the encoders clocks are continuously synchronised by a clock synchronisation protocol.
Figure 4: The implementation, exemplified with two encoders. Italic text indicates that the module has been modified in the work with this thesis.
3.2 Synchronisation

The synchronisation logic that is described in this section is located at the synchronisation module, shown in Figure 4. The module communicates with the decoder with an inter process communication protocol (IPC), and reads the frame buffer with another IPC protocol.

It is very likely that the frames in different frame buffers have been written at different times. When calculating a time difference between streams the different write times to the frame buffer needs to be taken into account. Not taking this into account may lead to an unnecessary big difference between streams, and also a kind of flip-flop behaviour where the frames are dropped or duplicated too often. This behaviour will degrade the QoE for the user, because the dropped or duplicated frame might be observed. To take into account the different write times, the writer (mpeg decoder in Figure 4) also attaches a timestamp of when the frame was written. Thus two timestamps are written to the frame buffer: the capturing time of the frame produced by the encoder and the write time of the frame into the frame buffer.

The compensation is done by taking another timestamp when the synchronisation calculation is going to be made \((ST)\), calculating the difference to the writing timestamp \((WT)\) and then adding the difference to the capturing timestamp \((CT)\). This calculation is shown in Equation 3.1.

\[
T = CT + (ST - WT)
\]  

When this calculation is done and the writing time has been taken into account, it is possible to compare timestamps from different frame buffers. However, some samples of these timestamps are taken first. The default value is 5 samples per second. When enough data has been gathered, it is time to calculate the average difference in time between the streams \((\delta_t)\). The calculation is shown in Equation 3.2 where \(T_{1i}\) represents the \(i^{th}\) timestamp from stream 1 and \(N\) represents total number of samples.

\[
\delta_t = \frac{\sum T_{1i} - T_{2i}}{N}
\]  

The equation only shows how to calculate time difference between two streams, if more than two streams are present the calculation is always done towards the slowest stream. Where the slowest is the stream with the earliest timestamps. When the \(\delta_t\) has been calculated, it is possible to translate this time difference into a difference in frames \((\delta_f)\). This calculation is shown in Equation 3.3 where FPS is the output frames per second (fps) and not the input fps of any of the streams. This is because the input fps of the streams might differ and it is the output synchronisation that matter.

\[
\delta_f = \frac{\delta_t}{ FPS}
\]

When the frame difference between all streams have been calculated, it is time to actually do the synchronisation. First of all if \(|\delta_f|\) is less than 1, nothing is done because the streams are considered synchronised. However a \(|\delta_f|\) greater than 1 indicates that the streams are unsynchronised and action needs to be taken. There are two actions that may be taken:
Table 2: NTP servers used to synchronise time against.

<table>
<thead>
<tr>
<th>name</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.se.pool.ntp.org</td>
<td>Sweden</td>
</tr>
<tr>
<td>1.se.pool.ntp.org</td>
<td>Sweden</td>
</tr>
<tr>
<td>2.se.pool.ntp.org</td>
<td>Sweden</td>
</tr>
<tr>
<td>3.se.pool.ntp.org</td>
<td>Sweden</td>
</tr>
<tr>
<td>svl1.ntp.se</td>
<td>Sweden (Sundsvall)</td>
</tr>
<tr>
<td>svl2.ntp.se</td>
<td>Sweden (Sundsvall)</td>
</tr>
</tbody>
</table>

1. Drop $|\delta_f|$ frames and decrease the video buffer by size of $|\delta_f|$ frames.
2. Duplicate $|\delta_f|$ frames and increase the video buffer by size of $|\delta_f|$ frames.

Note that it is possible to perform both actions at different decoders that will neutralise the same $|\delta_f|$. Either a speed up of the slowest decoder could be performed, or a slowdown of the fastest. Because of this freedom, the algorithm tries to minimise the size of the video buffers. However the buffers need to be kept at reasonable high levels to be able to fill their intended function. This results in that the algorithm always chooses to speed up the slowest decoder if the size of its video buffer is greater than the starting size of its video buffer. Thus it always tries to minimise the size of the video buffers and they are not shrunken too much.

### 3.3 Clock synchronisation protocols

This section describes which implementations of the clock synchronising protocols that were used. Obviously they were not used at the same time.

#### 3.3.1 NTP

The implementation uses the NTPD implementation of the NTP protocol, when NTP is going to be tested. It was chosen because it is the reference implementation of NTP, which gives it much credibility. NTPD implements version 4 of NTP, but it is also backwards compatible with version 1, 2 and 3. It is easy to configure and runs as a daemon in the background and continuously synchronises the computers clocks.

The NTP servers used can be observed in Table 2. Note that all are located in Sweden. This is to get as good time synchronisation as possible, since the experiments are done in Sweden. Especially close is the servers in Sundsvall which is about 300 km from the testing location in Umeå. Also the servers with “pool” in the name are not actually servers, they are a pool of multiple servers. When a connection is done to a pool, a server returned to the one connecting. Which can be used to synchronise time again. There is no guarantee that the same server will be returned every time.
3.3.2 PTP

As PTP protocol, the Linux PTP project has been used. It is a standard implementation that works well on Linux. Setting it up is easy, if all dependent kernel modules exist. If not, the kernel needs to be rebuilt and suddenly setting it up is not so easy. The PTP implementation used hardware timestamping.
4 Method

To evaluate the clock synchronisation protocols NTP and PTP, four tests have been performed. First, two tests that evaluated the implemented inter-sender synchronisation was done. These are used as reference tests, which represents the best results that could be performed by the inter-sender synchronisation implementation. The next test tested how NTP performed as clock synchronisation protocol between encoders and the last tested the same for PTP.

In all these four tests, frame difference at presentation time ($\delta_f$) was measured. Frame difference at presentation time was chosen because that is what matters to the end users. This is what the user perceives when synchronisation does not work correctly. Furthermore it can be used to calculate the offset in time between two streams. This is done by multiplying $\delta_f$ with the display time of the frame (i.e. one divided by fps). By doing this the results can be compared to studies regarding human perception of synchronisation. Thus, it is possible to draw conclusions about if the user will notice a specific value of $\delta_f$.

To measure $\delta_f$, information is inserted in the video stream. More specifically, a monotonically increasing frame number is inserted in the video data. This is done by representing a binary number with boxes of pixels that have the same colour (black represents 0, white 1) within the video frame. With this frame number it is possible to compare two streams, to see how far offset they are to each other. Doing this introduces a constraint that both streams has to originate from the same source, so they have the same frame numbers. Which makes it not a perfect model of reality, since in reality the encoders will have different sources that also may drift in time. Even though it is not a perfect model, it works for this case since the comparison focuses on the clock synchronisation protocols and not how the inter-sender synchronisation handles source drift.

All four tests had some common characteristics, they were the following:

- Encoders used were of the model Intinor Direkt Link 400.
- The decoder used was of the model Intinor Direkt Receiver.
- Both encoders and the decoder used a modified direkt firmware version 4.0.1. This modification is described in the implementation chapter.
- Video streams transmitted were encoded in H.264 (MPEG-4), with resolution 720p, a frame rate of 50 fps and with a bit rate of 10 Mbit/s.
- Tests were run for 20 hours.
- UDP was used as transport protocol for transporting MPEG-TS.
- One of the streams had a 2 seconds delay output buffer.
4.1 Reference tests

These two test were performed by streaming two video streams from the same encoder to a decoder. In a clock synchronisation perspective this is optimal, because both streams uses the exact same physical clock for timestamping. Thus, it serves as an indication on how the inter-sender synchronisation algorithm performs, without taking the clock synchronisation into account. The results of these tests can later be used to relate the other results to. They also provide insight on what may need improvement on the implementation.

The difference between the reference tests were when the source stream was split up into two separate and identical streams. In the first test it was delayed as long as possible. This was done by passing a single stream from the source to the encoder. When the stream is in the frame buffer, two encoder instances reads the buffer separately. These encoder instances encodes the frame, timestamps it, packages it into MPEG-TS and sends it to the decoder. One of the streams is held for two seconds before it is sent to the decoder, to simulate some network delay. Note that this is done after timestamping, thus the timestamps are not skewed by this delay. The test setup of the first reference test can be seen in Figure 5a.

In the second reference test the stream was split up earlier, at the source to be exact. The source sends the same stream over two Serial Digital Interface (SDI) cables to the encoder. In the encoder the streams are handled as if they had no relation to each other. This is a more realistic test setup, since all encoders will have difference sources in reality and not have the ability to read from the same frame buffer. Introducing this change, compared to reference test 1, will showcase how the system jitter before timestamping affects the inter-sender synchronisation. The test setup for the second reference test can be observed in Figure 5b.

![Figure 5: Test setup for both reference tests.](image-url)
Figure 6: Test setup for the NTP and PTP tests. Red arrows indicates PTP communication and blue indicates NTP communication. PTP and NTP are not used at the same time.

When both streams have arrived at the decoder, the same procedure is performed. The decoder waits until it has enough data of each stream, it decodes the video streams, synchronises and presents them. Alongside the presentation a daemon reads both frame buffers, extracts the frame number and calculates $\delta_f$.

4.2 NTP and PTP test

These two tests aims to evaluate how the clock synchronisation protocol affects the inter-sender synchronisation. Both test setups are very similar and they can be viewed in Figure 6. The only thing that differentiates them is what clock synchronisation protocol is in use. As in the second reference test, the source sends the streams through two SDI cables. However in these tests the cables are connected to two different encoders. The encoders does their thing encodes, timestamps and transmits the streams to the decoder. At the decoder nothing is changed, the same procedure is done and $\delta_f$ is calculated.
5 Results

This chapter showcases the results obtained from running the experiments described in Chapter 4. The results are displayed in histograms showcasing how $\delta_f$ is distributed in different values. These histograms display $|\delta_f|$ because a negative value is equal to the corresponding positive value, the only difference is which stream is ahead and which is behind. Also average values are included, to make the comparison between tests more straightforward. Remember that $\delta_f$ is a discrete variable, thus values are exactly the value which the staple stands on.

![Figure 7: Histogram of the outcome of the first reference test. The test had an average $|\delta_f|$ of 0.29](image)

The first reference test had mostly $\delta_f$ values at 0, and almost 30% with a $\delta_f$ of 1. Some values of $\delta_f$ can be observed at two, however when the difference became this big it was quickly corrected by the synchronisation implementation. The test run had an average $\delta_f$ of 0.29. The results of this test can be seen in Figure 7.

The second reference test is shown in Figure 8. It provided a shift towards a more unstable synchronisation. About 80% of all frames measured had either a $\delta_f$ of 0 or 1. A bigger presence of $\delta_f$ values at 2 was introduced. Even $\delta_f$ values of 3 was observed, although it was low. The average $\delta_f$ was 0.75 during this test.
Figure 8: Histogram of the outcome of the second test. The test had an average $|\delta_f|$ of 0.75.

Figure 9 shows the outcome of the NTP test. The results are centred around a $\delta_f$ value around 2, with 2 having about 40% of the outcome. Results also shows that the average $\delta_f$ was 1.9, close to the previous mentioned 2. Values as big as 4 are observed. This kind of size on the values starts to be problematic, but more about that in the next chapter.

The last test tested PTP, and is shown in Figure 10. This test it centred around a $\delta_f$ around 1 and 2, these two values represents about 65% of the outcome. An average value of $\delta_f$ was measured to 1.7. As in the NTP test, $\delta_f$ values as big as 4 was observed with about 5% of the outcome.
Figure 9: Histogram of the outcome of the NTP test. The test had an average $|\delta_f|$ of 1.9.

Figure 10: Histogram of the outcome of the PTP test. The test had an average $|\delta_f|$ of 1.7.
6 Discussion

This chapter discusses the results, practical implications of the protocols mentioned in this thesis and the GPS failure that occurred during the thesis.

6.1 Results

To be able to relate a $\delta_f$ value to the research in human perception of asynchrony, it has to be translated into time. This is done by taking the display time of the frame and multiplying it with $\delta_f$. The fps of the transmitted video in the tests was 50, this implicates a display time of 20 ms for each frame. Which means, for instance, that a $\delta_f$ value of 2 will produce a time difference of 40 ms. In the theory section, it was stated that the thresholds for humans to notice asynchrony is 80 ms. This correlates to a $\delta_f$ value of 4. Thus, values 4 and above will be considered as unacceptable and values below 4 as acceptable in a synchronisation perspective.

6.1.1 Implementation

Observing the results provided from the first reference test, the implementation seems to work excellent. Having $\delta_f$ values at 0 and 1, means that a user will never be able to notice asynchrony. Looking at the results of the second reference test, which is more adjusted to reality, the results become worse. Although the results are still within an acceptable level, since no $\delta_f$ value greater than 3 can be observed. The average values rises from 0.29 to 0.75, which is more than twice as high.

This difference highlights the impact system jitter has on synchronisation. Because in the first reference test, the stream is exactly the same until they are timestamped. Which means that the system jitter is very low and almost not present. However in the seconds reference test, the streams are divided early and timestamped long after the split has taken place. It could be as bad as $\delta_f$ is already 1 before timestamping, which automatically increments the final $\delta_f$ by one. It is likely that this is the case, because the results where heavily skewed towards negative values, almost never going positive. However this cannot be observed in the histograms, because the absolute value of $\delta_f$ is displayed.

6.1.2 Clock synchronisation protocols

Looking at the tests performed with the clock synchronisation protocols, it is possible to see that both operates at an acceptable level about 95% of the time. This is not good enough, since the user will notice asynchrony 5% of the time. Preferably $\delta_f$ should always be within the acceptable limit. NTP had an average $\delta_f$ of 1.9 and PTP 1.7, which is about one whole frame increase compared to reference test 2. This difference can only be explained with the
introduction of two encoders, and thus two physical clocks.

The average $\delta_f$ between the two protocols is not very big, 0.2 $\delta_f$. Both have about the same amount above the acceptable threshold, but PTP keeps $\delta_f$ at 1 more consistently than NTP. Even though PTP has a lower average it is not the most relevant aspect, it is more important to stay under the acceptable threshold. This is because a user will not notice the difference between a $\delta_f$ of 1 and 2. Thus a protocol that has an average $\delta_f$ of 3 and never reaching 4 would be better than a protocol with an average $\delta_f$ of 1 that are above 4 for some periods. Although this constraint exist, the protocol should always strive to have an average $\delta_f$ of 0.

These results indicates that the clock synchronisation protocol has little impact on the quality of the inter-sender synchronisation. The actual inter-sender synchronisation algorithm is much more important. Reducing system jitter before timestamping is probably most important to increase the quality of the inter-sender synchronisation, as shown by the reference tests.

With this information provided by the tests, it is possible to do an educated guess on how GPS would perform in this environment. Since PTP used hardware timestamping, it could achieve a clock synchronisation of about 100 nanoseconds. This is about the same as would be expected by a GPS that is set up right. Thus the results should most likely be closely related to the results of PTP.

### 6.2 Practical implications of clock synchronising protocols

Choosing clock synchronising protocol does not only depend on the performance showcased in this thesis. Performance may be better or worse in different set ups and/or environments. Protocols also introduce constraints that may not be possible to fulfil, and thus they are not able to be used in a specific case. Some light will be shed on what these constraints are and what environments each protocol used in this thesis thrives in.

#### 6.2.1 Hardware

To use GPS, an external GPS receiver is needed. Not any GPS receiver will do, special GPS receivers that has Pulse Per Second (PPS) functionality is needed. Also the interface in which the receiver talks to the computer will need to have PPS support. The receivers introduces an additional cost and it may be expensive, especially if many nodes needs to synchronise their clocks. Because every node will need a GPS receiver. PTP is not dependent on external hardware. However to get the best performance, hardware timestamping functionality is needed in the network card. If this is not available, the network cards needs to be replaced which is also implicates a cost. NTP does not require any specific hardware to work.

#### 6.2.2 Placement

The GPS receivers needs to be in line of sight with at least four satellites to work. This results in that the GPS receiver needs to be outdoors. Placing the receiver outdoors will probably introduce extra wiring of cables, since generally computers are located indoors. PTP introduces another placement constraint, this constraint being that all encoders needs
to be located on the same LAN. This also means that PTP is the only protocol that forces the encoders to know about each other. This is a flaw because it gets harder to introduce more encoders in the sending process, since all encoders in the network needs to be notified and not just the decoder. NTP does not introduce any big problems when choosing placement, the only thing is that the NTP servers should be as close as possible to the encoders. This is solved by switching NTP servers to a closer one, instead of moving the equipment.

6.2.3 Other constraints

NTP as well as PTP performs bad when asymmetric network delays are present. Since PTP is used on LANs, the network delay asymmetry will be less prominent than for NTP that is used in the internet. Because on the internet packages travels further as well as they share medium with a lot more competing packages that might cause the package to travel other routes. Introducing different routes might cause a difference in travelling time and thus an asymmetry in the network delay may be caused.

6.3 GPS failure

Initially the receivers was tested briefly, and they seemed to work fine. However no exact measurements were performed, which should have been done. Problems then occurred when time needed to be synchronised accurately in the tests. The GPS performed bad. Accuracy achieved was horrendous, even worse than NTP, which it should beat with a factor of at least 100. Having a GPS that performed worse than NTP would not be representative, and thus it was excluded from the experiments. Because it was so late in the thesis, there was no time to obtain a more accurate GPS.

It turned out that it was not the GPS receivers themselves that was the biggest problem, in fact it was the interface in which they communicated to the computer, Universal Serial Bus (USB). USB does a great job of standardising the connection of computer peripherals, such as keyboards etc. It is not ideal for tasks that is highly dependent on timings, such as clock synchronisation. The problem originates from the fact that USB is a polling protocol. Which means that the host polls each slot for data at a specified interval. USB will poll every slot once every millisecond. This may sound often, but it is not often when the clock synchronisation protocol tries to achieve a synchronisation on a sub millisecond level. [4]

It is important that the computer gets the time information as soon as possible, because it gets old for every nanosecond it is not handled. To provide information as fast as possible an interface that supports processor interrupts needs to be used e.g. RS-232. Also the GPS receiver needs to be able to send these interrupt signals, and this means that the GPS needs PPS support.
7 Conclusion and future work

An rudimentary inter-sender synchronisation algorithm has been implemented. The implementation is based on Intinors hardware and software, it uses MPEG-TS to transport streams and timestamps are inserted strategically in the MPEG-TS. Synchronisation is done by comparing timestamps at presentation time, and if the difference is too big frames are either dropped or duplicated.

The implementation has been tested with both NTP and PTP as clock synchronisation protocols on the encoders. Results from these tests indicates that the choice of clock synchronisation protocol is not especially important in a performance perspective when performing inter-sender synchronisation. However, if performance is extremely important it is possible to improve it a very small bit by choosing a better clock synchronisation protocol. Instead of performance, the clock synchronisation protocol should be chosen based on what constraints it introduces to the system. Thus it is important to know what these constrains are and what implications they have for the entire system. The protocols perform differently in different environments, which is also important to take into consideration. PTP, for example, might not be the best protocol to use if encoders need to be mobile and communication is done with 3G.

7.1 Future work

The experiments performed in this thesis have used only ethernet connections on LANs. This is the most common and a very good alternative. It would be interesting to see how the inter-sender synchronisation is affected by changing transport medium to, for instance, WiFi or 3G. This change would likely make PTP not a viable option, since it heavily depends on the ethernet connection. Thus this comparison would focus mostly on NTP and GPS. NTP might perform worse due to the worse transport medium and potentially added asymmetric delays. GPS will not be affected at all by this change, since it does not use the internet connection.

The conclusion of this thesis indicates that it is the synchronisation algorithm that dictates how good synchronisation that can be achieved, and not the clock synchronisation protocol. There exists comparisons of different synchronisation algorithms, for instance Ishibashi et al. in [6] and Boronat et al. in [3]. These comparisons focus mostly on features of the algorithms, and not the performance. Thus there exists a gap where performance needs to be evaluated for these inter-sender synchronisation protocols. This is a very big work, since each algorithm will need to be implemented and tested.

Also it would be interesting to see inter-sender synchronisation performance of a capture device that timestamps its created MF’s at capture time. This would be optimal, and it would remove all or most of the system jitter introduced before timestamping and thus provide
better synchronisation. However this will introduce a constrain that all capture devices will need to have synchronised clocks, which might be too great of a constraint compared to the benefits of it.
References


