MASTER'S THESIS | LUND UNIVERSITY 2015

Video Conferencing - Session and Transmission Control

Marcus Carlberg, Christoffer Stengren

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2015-04

Video Conferencing - Session and Transmission Control

Marcus Carlberg
carlber.marcus@gmail.com

Christoffer Stengren
christoffer_stengren@hotmail.com

February 2, 2015

Master's thesis work carried out at AXIS Communications AB.

Supervisor: Jimmy Rubin, jimmy.rubin@axis.com

Examiner: Mathias Haage, Mathias.Haage@cs.lth.se

Abstract

Video conferencing is a very bandwidth sensitive application, if the available bandwidth is to low to handle the send rate of the media, packages will be lost in the network and thus the conference will be disrupted. Axis Communications would like to know which techniques are used today to ensure optimized bandwidth usage and as good quality as possible during a video conference even if the network bandwidth changes. An implementation of such a service has been made in this thesis which uses three different TCP congestion avoidance algorithms. They monitor and evaluate the network quality to adapt the video stream rate accordingly. One algorithm is only based on packet loss and is used as a baseline. The other two uses packet loss in conjunction with round trip time (RTT) to evaluate the network. The user experience was deemed better when using the algorithms. The algorithms that was based on both packet loss and RTT was deemed superior. There are however still a few things to adapt in the camera software and hardware before a complete system can be developed.

Keywords: Video conference, Media streaming, Network congestion control, h.264, TCP-CUBIC, TCP-CUBIC-FIT, TCP-Illinois, GStreamer.

Acknowledgements

First we would like to thank Marcus Göransson for giving us the opportunity to write this masters thesis at Axis Communications AB. We would also like to thank our supervisor Jimmy Rubin for helping us to move forward when we got stuck and provide an open ear whenever we wanted feedback on any ideas. Further we would like to thank Robin Palmblad for helping us understand how the system works and the initial setup. We also give our thanks to Mathias Haage for helping us decide in which direction we should take the thesis. Finally a big thanks to all Axis personnel that we came in touch with and helped us out when we did run into difficulties.

Contents

1 Introduction **11**
- 1.1 Axis Communications . . . 11
- 1.2 Purpose and Goals . . . 11
- 1.3 Hypothesis . . . 12
- 1.4 Contribution . . . 12
- 1.5 Overview . . . 12

2 Background and Previous work **13**
- 2.1 Video conferencing . . . 13
  - 2.1.1 UDP . . . 13
  - 2.1.2 TCP . . . 13
  - 2.1.3 H.264 . . . 14
  - 2.1.4 Jitter . . . 15
  - 2.1.5 Packet loss . . . 15
  - 2.1.6 Round Trip Time . . . 15
  - 2.1.7 RTP . . . 16
  - 2.1.8 RTSP . . . 16
  - 2.1.9 RTCP . . . 16
- 2.2 TCP Congestion control . . . 18
  - 2.2.1 Loss-based . . . 18
  - 2.2.2 Delay-based . . . 18
  - 2.2.3 Hybrid . . . 18
- 2.3 Previous Work . . . 19
  - 2.3.1 TCP congestion control . . . 19
  - 2.3.2 Quality . . . 21
- 2.4 Limitations . . . 21
  - 2.4.1 Hardware limitations . . . 21
  - 2.4.2 Software limitations . . . 21

3 Experimental setup **23**
3.1 Hardware . . . 23
3.2 Software . . . 23
3.2.1 Rate Controller . . . 24

4 GStreamer **25**
4.1 Basics . . . 25
4.2 Pads . . . 26
4.3 GStreamer elements . . . 26
4.3.1 Source elements . . . 26
4.3.2 Sink elements . . . 26
4.3.3 Other types elements . . . 27
4.3.4 Data flow . . . 27
4.4 GStreamer bins . . . 27
4.5 GStreamer pipeline . . . 27

5 Audio **29**
5.1 G.711 . . . 29
5.2 Audio Synchronization . . . 30
5.3 Implementation . . . 30

6 TCP congestion control algorithms **31**
6.1 Motivation of Algorithms . . . 31
6.2 TCP-Illinois . . . 32
6.2.1 The algorithm . . . 32
6.2.2 Implementation . . . 33
6.3 TCP-CUBIC-FIT . . . 34
6.3.1 TCP-CUBIC . . . 34
6.3.2 TCP-FIT . . . 35
6.3.3 Combining TCP-CUBIC and TCP-FIT . . . 35
6.3.4 Implementation . . . 35

7 Evaluation **37**
7.1 QoS . . . 37
7.1.1 Definition . . . 37
7.1.2 QoS in video conferencing . . . 37
7.2 QoE . . . 38
7.2.1 QoE in video conferencing . . . 38
7.2.2 Measurment of QoE . . . 39
7.3 Limiting bit rate . . . 39
7.4 Approach . . . 39
7.5 Resulting measurements . . . 40
7.5.1 AXIS . . . 41
7.5.2 AXIS-PLL . . . 46
7.5.3 Lund . . . 48
7.5.4 USA peak hours . . . 51
7.5.5 USA low hours . . . 56

7.6 Packet loss . . . 60
7.6.1 AXIS . . . 60
7.6.2 AXIS-PLL . . . 61
7.6.3 Lund . . . 61
7.6.4 USA peak hours . . . 61
7.6.5 USA low peak hours . . . 62
7.7 Hardware Performance . . . 62
7.8 Evaluation process . . . 63

8 Discussion **65**
8.1 Local network tests . . . 65
8.1.1 Movement . . . 66
8.1.2 Alternating . . . 67
8.1.3 One Hour . . . 69
8.1.4 Video conference . . . 70
8.1.5 Jitter . . . 71
8.2 Lund tests . . . 71
8.3 USA tests . . . 72
8.4 Reducing the fluctuating pattern . . . 75
8.5 Performance . . . 75
8.6 Future work . . . 76
8.7 Conclusion . . . 77

Bibliography **79**

Appendix A GStreamer launch examples **87**

Appendix B Sample code **89**

Nomenclature

- **CBR** - Constant Bit Rate, bit rate that is constant
- **FEC** - Forward error correction.
- **GOP** - Group of Pictures
- **PLL** - Packet Loss Limit
- **QoE** - Quality of Experience
- **QoS** - Quality of Service
- **QP** - Quantization Parameter
- **RTCP** - Real time Control Protocol
- **RTP** - Real time Transport Protocol
- **RTSP** - Real time Streaming Protocol
- **RTT** - Round Trip Time
- **SVC** - Scalable video coding
- **TCP** - Transmission Control Protocol
- **UDP** - User Datagram Protocol

Chapter 1
Introduction

In this chapter we will first present Axis Communications AB where this master thesis was done. We will also go through the purpose and goals for the thesis and present our hypothesis and also present the distribution of work between the two authors.

1.1 Axis Communications

The masters thesis work is carried out at Axis Communications AB which is a company that specializes in developing and producing new and modern high performing network cameras. Axis Communications AB was founded in 1984 and has since then become the global market leader in network cameras. They were the first company in the world to release a network camera in 1996 and they have currently more than 1600 employees around the world [1].

1.2 Purpose and Goals

The purpose of this thesis is to investigate how we can use two Axis network cameras together to create a video conferencing application. The application will stream video and audio in real-time between the two cameras. The application shall support state of the art algorithms for maximizing the perceived quality for the users. Another master thesis has already been able to establish a video link between the two cameras and implemented some congestion control. Our task is to continue on this work and add audio to the video conference and make sure the audio is synchronized with the video. We will also continue to look at how we can counter problems such as varying bandwidth, jitter and loss of packets with the help of the literature. An implementation and testing of our finding will then be conducted.

1.3 Hypothesis

We believe that it should be possible to set up a video conference call transatlantic between two Axis network cameras where the QoE is rated high by both parties of the conference call. The connection is going to be TCP based while the transmission is going to be UDP based and use one of the hybrid algorithms to adjust the quality and rate for the media. If a hybrid algorithm was not to be used, we believe delay based is the second best option if no greedy algorithm is competing. If this is the case however, we believe a packet loss based algorithm would be the second best choice.

1.4 Contribution

This section presents what tasks the different authors had the main responsibility in.

- Marcus Carlberg - TCP-CUBIC, TCP-FIT, TCP-CUBIC-FIT, Introduction, Background and Previous work, Evaluation
- Christoffer Stengren - TCP-Illinois, Experimental setup, GStreamer, Discussion
- Marcus Carlberg - Implementation of TCP-Cubic, TCP-Fit, TCP-Cubic-Fit, video_client video and audio together, updated with newer software
- Christoffer Stengren - Implementation of TCP-Illinois, video_client single audio, single video, rate controller modifications

All other work was done in collaboration.

1.5 Overview

Chapter 2 will explain the basics of a video conference and what different protocols that are used to set up the different media streams. Chapter 2 will also explain what TCP congestion control is and how it has been used solving similar network problems. Chapter 3 will present the environment that we are working in, what hardware and software that we have available for the video conference. The streaming library GStreamer is explained in chapter 4 where we will take up the basics of what we used when implementing our conference application. In chapter 5 we present how we set up the audio stream and which protocols we used. Chapter 6 goes into detail on which congestion control algorithms we did choose and also shows how we implemented these. Chapter 7 begins by defining QoS and QoE and they will then discuss in regards to video conferencing. The chapter also explains how we collected data, present the resulting data of the collection and how we evaluated it. Chapter 8 will be a discussion on what we found out and a reflection on what implications it might have. We will also give suggestions for future work and conclude our findings.

Chapter 2
Background and Previous work

We will in this chapter present background material of this master thesis. We will go through what has already been done in the topic of congestion control and quality switching on network streams. We will also explain how the most common protocols that are used in network streaming works and what properties they have.

2.1 Video conferencing

With global partnership increasing a cheap and effective way of having meetings is required [21]. Video conferencing is therefore getting more and more important in today's society. In this section we will explain relevant theory regarding video conferencing.

2.1.1 UDP

User datagram protocol (UDP) is a protocol that is used to send datagrams (also called packets), from a computer to another. It uses minimal protocol mechanism and thus provides no guarantees for packet delivery nor correct ordering of packet arrival. UDP requires that the underlying protocol is Internet Protocol (IP) [29, 47].

Having minimal protocol mechanism provides certain advantages as well as disadvantages. As an example, the fact that we do not have packet delivery guarantee makes it very useful for application that cannot wait for a packet to be resent if it is lost and instead needs to skip it. On the flip side, if an application really need all the packets, and/or the packets in order, UDP would not work unless you make your own modification and implementations.

2.1.2 TCP

Transmission control protocol (TCP) is a connection based protocol that are used to transfer data-packets over a network connection. The protocol is used in applications where

it is important that all data packets are received and that they are received in the correct order. This is guaranteed with a constant connection between the client and the server where each packet sent results in an ACK-sendback. If a packet is lost and thus no ACK is received the packet is retransmitted. Since TCP is a connection based protocol it has a lot of features that is unavailable in UDP. TCP can for instance detect errors in the transmission by comparing the checksum from the packet and the checksum in the ACK that the client responds with [30, 46]. Another example is congestion control, algorithms that are designed to check the network for congestion and notify the server of an incoming congestion so the server has time to adapt. As TCP has a guarantee that all data packet will be received, it is possible to get reliable network statistics for congestion control. This is hard with UDP as you have no way of getting reliable statistics due to not having this guarantee.

2.1.3 H.264

H.264 is currently one of the most popular video compression format. It is used for recording, compression and distribution of videos. It was developed by ITU-T Video Coding Experts Group and is used in many products such as streaming services (youtube, vimeo, itunes) and web software(Adobe Flash Player, Microsoft Silverlight). H.264 provides many useful features such as profiles, levels, SVC, error and loss concealment [6].

Profiles defines specific capabilities that can be used for different applications. Example from wikipedia [6] "Extended Profile (XP) - Intended as the streaming video profile, this profile has relatively high compression capability and some extra tricks for robustness to data losses and server stream switching." Profiles are with other words very convenient as you can choose which profile fits your application [6, 56].

Levels defines specific performances on the decoder. This lets you choose a level that fits your application. Each level determines things such as maximum picture resolution and bit rate. This gives you the flexibility to for example change compression bit rate depending on the bandwidth. It is important to know that if you choose a level the decoder needs to be able to decode any level below it [6, 56].

SVC is short for scalable video coding. SVC divides the media content in different layers that can be added together. Combined they give you the high quality image. The different layers are dependent on each other, you must have all of the underlying layers to be able to reconstruct the stream. This enables you to send different bit streams at the same time. If however you get low on bandwidth you have the option to drop any of the bit streams. This yields lower quality images but reduces the bandwidth usage [6, 15, 57].

Loss resilience deals with the fact that losses may occur, such as a part of or an entire frame is lost. If that occurs and if it is possible, it is concealed by different methods so that a human will not notice or notice vaguely. It also provides different error/loss robustness technique [6, 58].

H.264 video stream

A typical video stream is a series of pictures that is shown in sequence. H.264 uses different types of image frames to try to reduce the size of the media stream. The first frame that it uses is called the I-frame and it contains a full image. The image is still compressed but

all data is available. When movement happens in a video stream it is often not the whole image that changes, but only a fraction. It is therefore unnecessary to use only I frames. H.264 uses this information to include two types of partial frames, the P-frame and the B-frame. Both the P-frame and the B-frame contains only the differences (the movement that has happened since the last frame) from the previous I-frame. One could say that the P and B frames updates the I frame. By using this technique the data necessary for showing the stream is reduced greatly without the quality changing noteworthy. However the P and B frames can't be decoded without the I-frame, this results in that all P and B frames gets invalid if an I-frame is lost. Therefore I-frames are sent in regular intervals to make sure that the stream is not unavailable for a long time because an I-frame was lost. The difference between the P and B frame is that P frames use data from the previous frames whiles the B frame can use data from both previous and the next frame in the stream. The B frame has more information available in the surrounding frames so it will not require as much data as an P frame. [17]

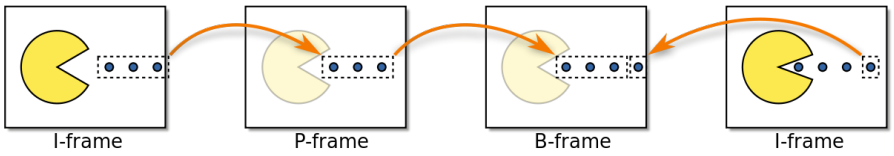

Figure 2.1: Illustration of how I-frames, P-frames and B-frames work together.

2.1.4 Jitter

Jitter is simply put the deviation between a signals that is expected to be periodic and its true periodic form. In our case this will be translated to packets that are presumed to arrive at a periodic interval but may have some deviations from this interval [23, 31]. In most video streaming applications jitter will become delay instead of jitter since most applications uses a image buffer and doesn't show the next frame until the buffer has reached a certain size [49].

2.1.5 Packet loss

When sending traffic over the internet it is possible that we lose packets during the transmission for one of many reasons. This is exactly what packet loss is, packets that have been lost for any reason and does not arrive at its intended destination [23].

2.1.6 Round Trip Time

The round trip time is the time it takes for a package to travel to its destination and back again. It is important to have a low RTT in videoconferencing since it will translate to delay in the call. This might cause problems when calls are made over a great physical distance potentially increasing network delay significantly.

2.1.7 RTP

RTP is a protocol that is the standard for transporting both audio and video media stream data over networks. It has to be used with UDP as transport protocol and thus has no guaranteed packet delivery [34]. However RTP has features for jitter compensation and packet reordering if packets arrives in the wrong order. RTP is a very versatile format and can handle a lot of different formats. The needs of the media type is not hard printed in the protocol but is instead provided as a profile in the RTP header. By using profiles RTP ensures that it can handle every new format that is presented on the market whether its a new video encoding or a new audio encoding. Everything that is needed is that the new format implements a profile that defines the codecs and payload to be used [13].

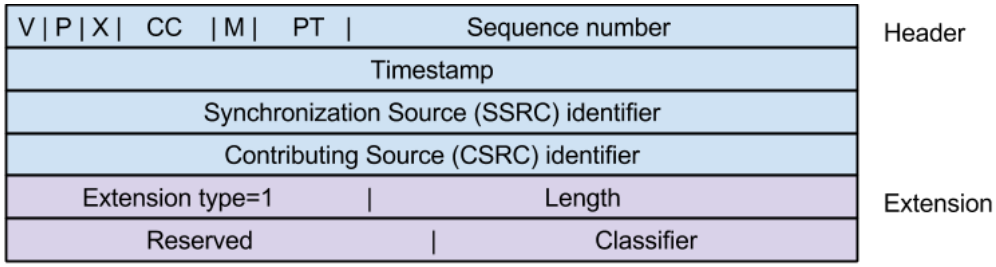

Figure 2.2: The structure of an RTP packet. Every bar is 32bit long. V stands for Version, P padding, X Extension, CC CSRC count, M marker and PT for payload type

2.1.8 RTSP

Real time streaming protocol is used to control the delivery of data with real time properties. It works in the application layer and is not concerned with the actual transmission. This must be done by and other protocol such as RTP. RTSP provides different means to control the delivery such as delivery via TCP or UDP, or delivery with RTP based mechanisms. It also lets you control the stream with commands such as play and pause [11] [12].

2.1.9 RTCP

The Real-time control protocol is an control protocol and is used for mainly four different purposes [14]. The first is to provide assessment about the data distribution and its quality. This is very important as it used in other protocol, such as congestion control protocols, and is considered its main function. In our congestion control implementations we will uses precisely this feedback, primarily round trip time and packet loss.

The second one is to keep track of an constant identifier for CNAME, and RTP source. If for instance an restart happens or a conflict occur, it may be so that other identifiers may change, and CNAME is needed to keep track on the participants and the identifier CNAME as it need to be constant. CNAME may also be used for synchronization between audio and video.

The third purpose is to control the send rate of packets. It is required that all participants sends RTCP packets to all others. That way each user can see how many participants there are and scale it sends rate according to how many there is.

The forth one is optional and is used to provide minimal session control. As an example the user may want to display all other participants identification in an user interface.

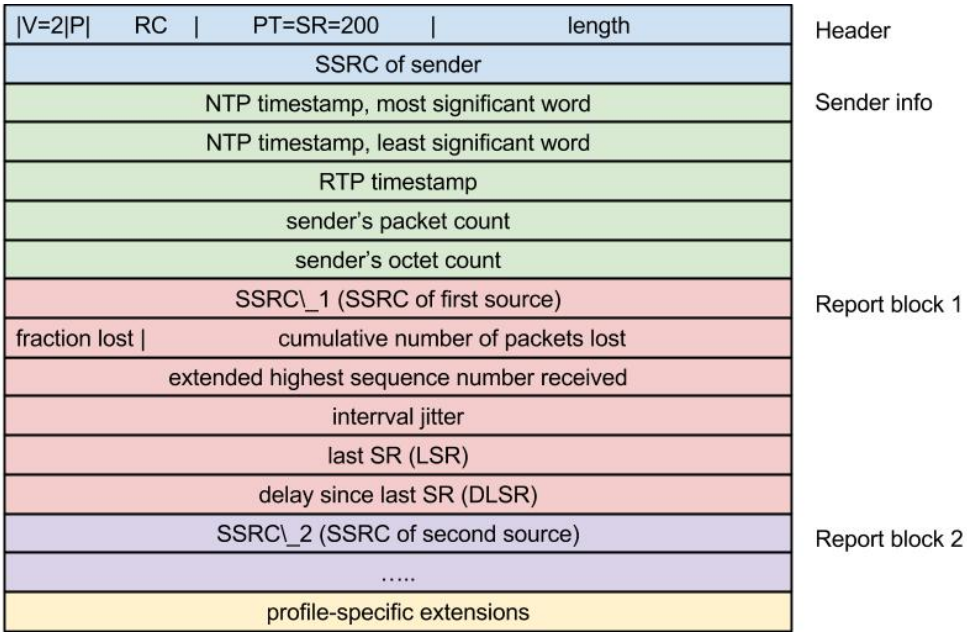

Figure 2.3: The structure of an RTCP packet. Every bar is 32bit long

2.2 TCP Congestion control

TCP congestion control is the topic of how to control the TCP flow to avoid congestion. Without it the internet would suffer a congestion collapse [18]. Congestion occurs as a result of two things, router queues are not infinitely long and TCP retransmits packets at packet loss. When a network is affected by heavy load, the routers don't have time to route all the traffic and thus the queue gets full. When the queue is full the packets that arrive at the router has no place to go and must be discarded. The sender of the discarded packet will not receive an acknowledge for that packet from the receiver and will then try to send the packet again. Every client on the congested network that uses TCP will do the same thing and thus the queue will never be empty. There are a dire need to control the congestion in TCP networks and this is done by different congestion control algorithms [23]. The way they work is by controlling the amount that should be sent. This is done by looking at something called the window size. The window represent how much it shall send. The size is then adjusted depending of how much it should and can send. This in turn depends on many things such as available bandwidth and fairness with regards of other algorithms running at the same time [19, 32, 33, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 59, 60]. There is currently three different types of TCP congestion control algorithms: Loss based, delay based and hybrids.

2.2.1 Loss-based

The loss based uses the loss of packets to determine how to control the size of the window. If there is a packet loss it will reduce the window size(most commonly by one half) and then slowly increase it with each ACK it receives from the destination. This process is iterated during the duration of the TCP session. Since loss based only reacts when a packet loss has occurred the congestion is not avoided but instead are handled after it has occurred. Another technique that is better in avoiding congestion all together is delay-based congestion control [32].

2.2.2 Delay-based

The delay based work by looking at the queue delay instead, also called round trip time (RTT). By looking at this we can see that when the delay get bigger, it probably is an indication that a congestion is soon imminent or is starting to build up. It will then reduce the window size in response. Similarly when the delay is small it will start to increase the window size but more slowly than it decreased it. Delay based can be seen to try to predict congestion and adjust its send rate accordingly [39].

2.2.3 Hybrid

The hybrids works by using both the delay and packet loss to determine its send rate. As an example one might use packet loss to determine if we are to decrease or increase the window size and use delay to see by how much. This is a method TCP-Illinois use [43].

2.3 Previous Work

Today streaming media like movies, television series and short clips takes up a large part of the total amount of internet traffic in Europe and America [51]. This is a lot due to services like `youtube.com` and `netflix.com` that streams recorded media content stored on a server to it's users. Since these kind of services takes up most of the bandwidth compared to other types of streaming [51] most research that we found were based on that model. The media in these services already exists on a hard drive and is therefore preprocessed and optimized for different qualities of the media. This optimization of the data files make it easy to adapt to sudden quality changes. These kind of services can have the stream delayed by a couple of seconds without the consumer even noticing it more than on the stream setup time. In video conferencing however the source comes directly from the camera in real time and therefore some optimizations are impossible or would require to much processing power to be able to do in real-time. Another problem with real time streams is the buffer size. When you stream a prerecorded video file the buffers can be several seconds big, but with real time streams the quality of experience (QoE) is drastically decreased when the delay is more than 150ms. For that reason the buffer can't be to large because the delay may become very large very fast [24] [54]. Since video conferencing has these requirements we had to focus on two subgroups of research. First how to identify and solve congestion in the network. Secondly how we can change the video stream in a way that the frames can fit in the new packet window size without the stream losing valuable information.

There do however already exist applications that can run a video conference. As a good example of this is Skype who is the market leader VoIP application worldwide [22]. We would have liked to look more at applications like these but unfortunately most of the protocols that Skype uses are encrypted and it's implementation is classified. However many studies has been done to try to identify what congestion control that Skype uses, one of them [22] has identified that Skype uses both the RTT and the loss rate as parameters when calculating the new window size. They did also found out that variations in the RTT does not impact the input rate a lot. They also mentions that Skype runs in three different states, the first state is when the network is stable and no congestion is detected. Skype uses CBR in this state. The next state is the congestion state which Skype enters when congestion has been found and uses it's congestion avoidance algorithm to prevent the congestion. The final state is the loss state. Skype enters this state when running on a lossy network where the loss is not due to congestion. Skype also uses FEC to minimize the losses in the network [22].

Skype starts its session with sending lots of packets with varying size for the first 20 seconds to determine what level of bandwidth it can use. It also reacts to packet loss by increasing the packet size and preferable uses UDP, but switches to TCP if for example a router does not accept UDP. [20]

2.3.1 TCP congestion control

A lot of research has been done on TCP congestion control. However most of the algorithms were only tested on applications that can handle delays in the size of seconds and might not guarantee that they are fast enough for a video conferencing application.

The algorithms will most likely function well or even very well nevertheless since many algorithms has been used as standards for multipurpose programs in different operating systems. The main research trends seems to be to have an algorithm that can hog as much bandwidth as possible and still be friendly to the most common known algorithms that are used. TCP-CUBIC-FIT is an example of this. It is an hybrid congestion control protocol that tries to remain friendly with TCP- CUBIC, that was the Linux standard for a while, and still have the benefits of being both loss based and delay based [36].

For the interested reader here are a couple of the most common algorithms that we have looked at in the different categories. Every algorithm below either tries to adapt to one or more of the other algorithms or are one of the most common used congestion control algorithms.

- Loss-based
  - PRR-TCP [45]
  - TCP CUBIC [32]
  - TCP BIC [42]
  - HighSpeed-TCP (HS-TCP) [44]
  - Scalable TCP [38]
  - TCP-Reno [60]
  - TCP-NewReno [35]
- Delay-based
  - TCP-Vegas [19]
  - FAST-TCP [59]
  - TCP-Westwood+ [33] [32]
  - TCP-AFRICA [39]
- Hybrid
  - TCP-Illinois [43]
  - TCP-Compound [40]
  - TCP CUBIC-FIT [36]
  - TCP-FIT [37]

The current standard congestion control algorithm for Windows server 2008 is TCP-Compound [8] and for Linux PRR-TCP [9].

2.3.2 Quality

Changing quality of an image in streaming can be done in many different ways. We have already talked about different bitstreams in SVC and how we can apply all of them to get the highest quality. Other ways is simply to compress the image to different degrees. It is also possible to use predefined profiles such as in H.264. Frame rates can be reduced to increase quality as will be explained in 3.6. It is also possible to use layered imaging [50]. In another article [55] they describe an quality adapting method that divides the video in different layers where the fist layer is the lowest quality and the sum of all layer is the highest quality. This is similar to SVC but instead of bit rates it sends different layers of the image instead. The receiver then use constraint programming for selecting which layer at which time that should be downloaded.

There are also a lot of research done to handle sudden shifts in bandwidth, one example of this is [41]. They use a deadline based protocol that sorts the most important frames(generally I-frames) in H.264 and send them first over TCP instead of sending the frames in playback order. The frames that are sent first are thus those that contain most information of the playback. Without these frames the playback would not make sense since we cannot decode the P-frames without it's corresponding I-frame. The less important frames are sent via UDP as they are not as important. A packet of sorted frames are then given a deadline, if not all of the frames are being transferred when the deadline is reached the frames are discarded and the next packet of frames can be sent.

2.4 Limitations

In this section we will list various limitations and constraints we had to work around.

2.4.1 Hardware limitations

The camera will be both the client and server. This means that the camera is responsible for recording, encoding, transfer, decoding and playback of video and audio resulting in that the available processing power is limited. In a real application you would like to show the video stream on for example a TV screen. In this thesis we instead recorded the video stream to an SD card for playback at a later stages.

2.4.2 Software limitations

Here we will list various software limitations we had in this thesis.

H.264

H.264 have some very useful features. At AXIS however we do not have access to all features but only a subsection of them to work with. To control the image compression we can use a variable called the QP value that lets us set the compression grade of the video stream. The modified H.264 format also gives us the possibility to set a maximum bit rate

and decide if it should prioritize frame rate or quality. It will then try to never go above this maximum by automatically regulate the compression grade.

Decoder

We didn't use the decoder as we didn't have a video output on our cameras. We instead choose to focus on other areas. This could affect the performance of the camera in several ways and need to be taken in account in future work. It is for example possible that adding a decoder might lead to a extra delay and desynchronization of the audio and video streams.

Chapter 3
Experimental setup

In this chapter we will go through the environment we worked with. We will explain the hardware setup and also mention any special software we where using.

3.1 Hardware

The hardware we used are two F41 network cameras. The F41 is a camera that can capture up to 1080p resolution using a camera head that is mounted on a cable. For the interested reader who want to read more about this camera we recommend this link: `http://www.axis.com/products/cam_f41/`. During this thesis we used different setups. The first one consist of the two cameras connected to a switch. A computer was also plugged into the switch so we could access the cameras. The second setup consisted of each camera being plugged into a different router. One computer was then plugged into one of the routers so that we could access one camera locally and one remotely. We also used a setup where only one camera was plugged into a router, the other one being directly plugged into the Ethernet wall socket.

3.2 Software

Our base software is the current available software that runs on the AXIS cameras. This helped us a lot as we did not need to implement things such as a own made rate controller. This thesis was a continuation of an previous master thesis. Because of this we used some of their solutions, such as how to get the RTCP statistics to our algorithms, making improvements as necessary. Though as we also used two network cameras with a new software base in it, we had to adapt and change a lot to be able to use those solutions. In this thesis we programmed our algorithm in C, and used Git as Configuration Management tool. To build our software we used make tools and gcc . To measure the performance on

the cameras we used a program called hog_client which was able to monitor the CPU and RAM-memory performance. To retrieve network statistics we used RTCP packets and printed the data to a file.

3.2.1 Rate Controller

Axis rate controller is a controller that is able to change both the quality of the picture and the frame rate of a video stream to adapt to a specified bit rate. The user of the camera can specify this target bit rate. If there are a lot of motion in the picture and thus more information has to be sent over the stream, the rate controller responds appropriately by lowering the frame rate or the quality of the frames sent.

There are some limitations however that we had to take in account. To cope with these we had to modify the behavior of the rate controller to make it more useful in a video conference purpose. The first problem encountered was that it is not possible to change the desired bit rate without restarting the stream which would result in a conference call running on the same quality for the whole duration of the call. The second problem was that if the network couldn't handle the specified bit rate, say the available bandwidth suddenly is halved, frames would be lost as it would send to large or too many packets. On ideal network links the rate controller works best if the bit rate doesn't sink below 300 kbps and not rise higher than 50 000 kbps (though even at max bit rate the bandwidth allocation seldom rises above 6 000 kbps when using H.264).

Chapter 4

GStreamer

In this chapter we will go through the basics of GStreamer which is the streaming library that is used on the AXIS network cameras. For the interested reader we recommend the official GStreamer manual[5] for more detail and as an excellent starting manual to GStreamer. If the reader is interested in coding examples there exist some in Appendix A. The manual [5] is also recommended for this purpose as well.

4.1 Basics

To get a basic understanding on how GStreamer functions we will first explain the fundamentals. We will then dig deeper on each of the topics in separate sections.

A fully functional GStreamer stream consists of a number of elements. The main one is the pipeline. The pipeline can be seen as a container of all the elements of the stream. If you want a video stream you connect elements of the appropriate types. For example if you have a H.264 file that you want to stream and save to a disk in another format, you would first add an element that contains the stream source say a H.264 file source. Then you may want to parse the stream and then you have to link the source to a parser. To decode it you would then link it to a decoder that can translate the file format to the data format that you want. Finally you have to link the decoded stream to a file element that writes to a file. All these elements are linked together by something called pads and then put into the pipeline where the stream can be set to playing. We will now go more deeper into how the different parts work and explain more thoroughly how they are meant to be used.

4.2 Pads

Pads can be described as an element that provide communication to the world outside [5]. There exist two kinds of these called source pads and sink pads. An element will use a source pad to output its data to for example another element. The sink pad works the opposite, an element will use that to receive data. Pads may have different kind of availabilities, always, sometimes and on request. They work just like they indicate, example on request only exist if requested to do so.

Figure 4.1: A source pad and a sink pad. The arrow represent the data flow between the two

4.3 GStreamer elements

GStreamer elements are objects that perform one specific function such as reading from a file, decoding a signal or sampling a signal [5]. This lets you as a developer easily create an object that represent one specific function. Various GStreamer elements already exist and it is possible to make your own or import others. It is possible to change the state of a GStreamer element. Say you want to pause an element, you simply change it state to the pause state. It is important to note that an element will not do anything until you have manually set its state to playing. GStreamer has a class GstElement representing an element. This is what all decoders, samplers, input and output functionality are classified as.

4.3.1 Source elements

Source elements are elements that have some form of source to generate data from [5]. We mentioned earlier that we wanted to stream a H.264 file. To read this file we would typically use a source element which could read and generate data from it. It is important to note that source elements does only generate data and does not accept any. It is with other word not possible to have a source element at the end of a chain of elements.

4.3.2 Sink elements

Sink elements are the opposite of source elements. They do not generate but only accepts data [5]. One example of this is an element that accepts data to output it to the standard audio output, like the element pulsesink. It is used to send an audio stream to the default audio output on your computer. Sink elements are used at the end of an element chain.

4.3.3 Other types elements

There exist a range of other types of different elements such as filters, converters, codecs and more [5]. They all work like a filter or in a filter like way. A filter have one sink pad and one source pad. They will receive data on their sink pad, do something with it, and then generate the modified data to its source sink. One example of this is a volume element where it accepts audio data, filters the volume, and then output it again. An example of elements that does not work quite as filters but in a filter like way is demuxers. Just as an filter element they have one sink pad, but instead of one they have multiple source pads.

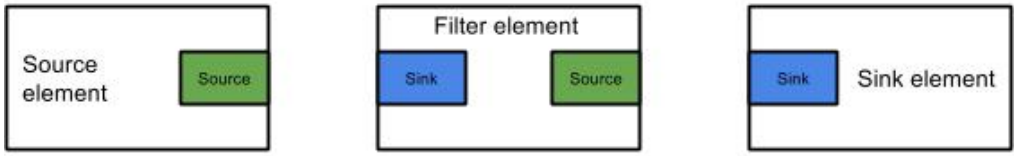

Figure 4.2: Three different kind of GStreamer elements, a source, a filter and a sink element

4.3.4 Data flow

One might be confused to why a source element would have a source pad and not a sink pad. Surly it would make more sense for it to have a sink pad to where it could "sink" its data, as a source element only generate data and a sink elements only accepts data? The other way around can be asked for a sink element.

The terms source and sink however is relative to the flow of the data [5]. In the perspective of pads, the flow will always go from a source pad to a sink pad. In the perspective of elements, the flow will always go from an source element to an sink element.

4.4 GStreamer bins

A bin is simply an container of GStreamer elements [5]. This is quite convenient as it lets you treat a bunch of elements as one. You could for example change the state of all elements in the bin at once to playing. This is due to a bin being a subclass of GStreamer element. A top level bin is called pipeline and can be looked at as an bin which collects several elements and bins and connects them from a source all the way to a sink element.

4.5 GStreamer pipeline

If you have many elements you may want to chain them together. As an example you may have a file that a reader elements shall read, a filter elements to do something with it and an output element that outputs it to the user. To chain these together a pipeline is created [5]. The elements are then added and linked together in this pipeline. This lets data flow

from the beginning of the pipeline to the end, going through all elements in it. A way to look at a pipeline is as the name suggest as a pipeline. Each element will make up a pipe in the pipeline and the data will go through all the different pipes in the direction from the source pipe to the sink pipe. A pipeline can be controlled by setting it to different states as it is a top level bin. If you want it to start you set its state to the playing state and will subsequently set the state of all elements in it to play as well. Likewise if you want it to pause you set it to the pause state. A pipeline will continue to run until data runs out or you stop it manually.

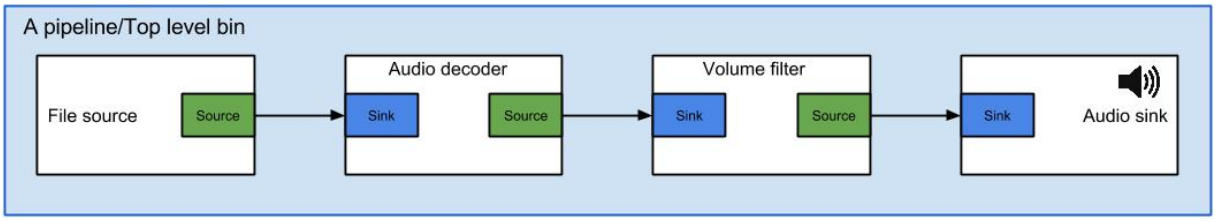

Figure 4.3: An simple audio pipeline. The arrows represent the data flow

Chapter 5
Audio

One important part for a successful conference is audio. In this chapter we will discuss how we implemented audio in the video conference application on the AXIS cameras and the importance of synchronization.

5.1 G.711

The audio encoding used in this thesis is the G.711 and it is an old audio encoding standard [4]. It was released as early as 1972 and we are using it as a first implementation of sound in the conference system because of it simplicity. G.711 uses a sampling rate of 8 khz and thus a stable bit rate of 64 kbps. There are several benefits of a simple protocol for us. First it is easier to use so we can start to work on synchronization and bandwidth allocation problems that arises with having two streams (one audio and one video) at the same time. The second part is that as we mentioned before that the camera processing power is not limitless and using a simple protocol which only runs on one bit rate and one sample rate will keep the cpu load constant during a call.

Another benefit is that nearly all G.7XX protocols has been used in telecommunication applications. Thereby we can be sure that the encoding protocol works well on the desired application [3].

Another audio format that should be better to use is OPUS [7] (harder to implement but a better codec overall). OPUS provides the user with a few features that comes in handy when handling video conferences like setting the audio quality while the stream is live and thus giving us more power when adjusting the rate controller by being able to alter the sound too and not just the video. The audio quality could also be increased by a large margin during conditions with very high bandwidth ranging from 6 to 510 kbps (nearly 8 times as many bits as G.711). Unfortunately there were no time to implement OPUS in this thesis.

5.2 Audio Synchronization

There exist many methods to handle audio synchronization. The most straight forward approach is to us time stamps on both the audio and video content and then wait for the audio and video segment with the same timestamp before showing the stream to the user. This approach would work fine, but lost packets from the different streams must be handled appropriately. The user would not want the stream to go down if either the audio or video is missing.

There are algorithms that synchronizes the audio based on lip movement, however these algorithms are very complex to implement and must often be told where the users mouth is to be accurate.

Another method is called CAVLC and it uses a special encoding were the audio information is encoded into the video segments. This will however lead to that if we loose a packet we will loose both the audio and the video information [48]. In our implementation the audio must be superior to the video, a conference can still continue without the video feed but without the audio this is not possible.

5.3 Implementation

We choose to implement audio synchronization with help of GStreamer inherit synchronization where timestamps are compared and then ordered accordingly to be played at the same time. Since we already worked with GStreamer it was deemed easiest to try and see how it worked before testing more advanced audio synchronization mechanisms. To be able to the synchronization in GStreamer the audio and video needs to be transferred in the same pipeline. This is easily done by just creating both the audio pipeline and the video pipeline and linking the pipelines individually and then adding them to the same pipeline. In GStreamer the audio has priority over the video when synchronization is concerned and if a video packet is lost the audio will still be played as usual.

Chapter 6
TCP congestion control algorithms

As mentioned before, TCP congestion control is an important part in network communication. We will compare different TCP congestion algorithms to each other to choose which one to implement. Finally we will discuss our implementation.

6.1 Motivation of Algorithms

The previous master thesis solution that uses TCP-Vegas have one big drawback. As [39] and [43] mention TCP-vegas cannot compete well with an ordinary TCP congestion control algorithm such as TCP-Reno. What happen is that TCP-vegas will only get a small portion of the link capacity as it will release the capacity to a greedy algorithm, such as TCP-Reno. FAST TCP sound nice, but as [43] mention it is not fair and adapts to TCP-Reno. Instead the bandwidth allocation between the two depends solely on which enters first.

The big drawback on only loss based algorithms is the fact that they are not able to utilize the full bandwidth potential it could actually use. Delay based does not have this problem and generally work around full utilization. The problem with them however are that they require accurate measurement of the queuing delay, and this might not always be possible [43].

Hybrid algorithms that uses both loss and delay tends to try to solve these problems by using the strengths of both. Some succeed better then others and as more and more research is done better and better hybrid algorithms has been made. For these reasons we believe that a hybrid algorithm is the best choice. We will thus implement two of those. We will also implement a pure loss based one and compare the results to the hybrid algorithms. An pure delay is already implemented.

TCP-Illinois and TCP-CUBIC-FIT are the leading hybrid algorithms that so far looks to be the most promising for us. Plain CUBIC is our candidate for implementation when it comes to loss based as it has been standard in Linux up to 3.2 where it was replaced by

PRR [9]. We choose to not implement PRR because we wanted to see how much benefit a loss based algorithm(CUBIC) gets when combined with a delay based modification(FIT).

6.2 TCP-Illinois

TCP-Illinois was introduced in 2007 as a new TCP congestion control algorithm using both delay and loss as its signal [43]. It was designed to be fair with other already existing algorithms and to take in account high speed networks. The algorithm uses an window size which it expand if there is no congestion and decrease it if there is congestion. This is done in a additive increment, multiplicative decrement manner. It has been noted to outperform TCP-Vegas which is the current existing implementation.

6.2.1 The algorithm

In order to determine if it should increase or decrease the window size the algorithm looks at packet loss. If we have a packet loss, it will decrease the window size. If instead an ACK is received from a successfully transmitted packet, it will increase the window size. This is the primary congestion signal it uses. The reason behind this is that measurements of RTT can be noisy which could decrease the performance. By instead using the packet loss as a primary control algorithm it gets more robust. Delay, however, is instead used as a secondary control signal to determine how much to increase and decrease the window size. Delay makes it possible to estimate if congestion is imminent by looking at the RTT. The higher RTT the more likely it is congestion is occurring. The algorithmen uses several parameters that all have standard settings in [43]. These include the following: $\alpha_{min}, \beta_{min}$, $\alpha_{max}, \beta_{max}, W_{tresh}, \eta_1, \eta_2, \eta_3$. The window size is defined as W. The first thing the algorithm do is to determine average RTT T_a, maximum average RTT T_{max} and minimum average RTT T_{min} over the last W acknowledgements. It will then calculate the maximum average queuing delay $d_m = T_{max}$ - T_{min} and the current average queuing delay $d_a = T_a$ - T_{min}. It then sets $d_i = \eta i \cdot d_m$. The algorithm proceeds to calculate two variables α and β according to

$$\alpha = \begin{cases} \alpha_{max} & d_a \leq d_1 \\ \dfrac{k_1}{(k_2 + d_a)} & otherwise \end{cases} \tag{6.1}$$

$$\beta = \begin{cases} \beta_{min} & d_a \leq d_2 \\ k_3 + k_4 d_a & d_2 < d_a < d_3 \\ \beta_{max} & otherwise \end{cases} \tag{6.2}$$

where

$$k_1 = \frac{(d_m - d_1)\alpha_{min}\alpha_{max})}{(\alpha_{max} - \alpha_{min})} \tag{6.3}$$

$$k_2 = \frac{(d_m - d_1)\alpha_{min}}{\alpha_{max} - \alpha_{min}} - d_1 \tag{6.4}$$

$$k_3 = \frac{\beta_{min} d_3 - \beta_{max} d_2}{d_3 - d_2} \tag{6.5}$$

$$k_4 = \frac{\beta_{max} - \beta_{min}}{d_3 - d_2} \tag{6.6}$$

The k_i values are then updated each time either T_{max} or T_{min} are changed so that these are always up to date. The α and β are updated on each RTT. So far only the secondary congestion signal has been used. Now the primary is used to determine how the windows size W should changed. If we received an ACK, W is increased with α/W. If a packet loss was detected due to triple ACK it is decreased with $\beta \cdot W$. Any time a packet is detected by a timeout, α is set to be 1, β to 1/2 and W threshold to be $W/2$.

One important part to the algorithm is when alpha is set we do not allow it to be set to α_{max} directly when $d_a > d_1$. This is to make sure that the windows size is reduced properly and not suddenly increased due to packet or measurements noise. In order to be set to α_{max}, d_1 needs to be smaller or equal to d_a for at least $\theta \cdot RTT$.

The behavior of the algorithm can be described as follows: When it first starts it will look at the RTT to determine if there is a congestion imminent or not. This will determine the the value of α and β which are used to change the size of the window. Due to how this change is done and how α and β is calculated, the window size will drastically be reduced if a package loss occur to ensure as good performance as possible. Directly after such a package loss it will start a slow recovery which gradually increases until the maximum windows size W_{tresh} is reached. This gradually increase is explained like this. In the beginning we are unsure if the congestion is actually over or not. Thus we increase it slowly to begin with just to be sure. This uncertainty can be for many reasons, such as a noise connection, and is an important step. If we have not received a packet loss for a while we assume the congestion is over and we increase the window size more sharply, as seen in equation 6.1 where we might even set it to the maximum.

6.2.2 Implementation

When implementing the algorithm some changes had to be done with it to fit our application. For one, we could not determine how a package loss was detected, only that it had happened. Due to this reason the parameter W was always decreased according to if a loss was detected with triple ACK's. The timeout was not implemented as we had no way of determine when one had occurred.

In the algorithm RTT interval and RTT is used, e.g. $\theta \cdot RTT$. However, our algorithm does not look at every RTT but only once every 0.5 to 1 second due to limitations. We instead used an increment counter and each time incrementing it with one, representing an RTT interval. If this was not done the algorithm would simply only look at one RTT, take it as the average and set both min and max to be that very same RTT making the algorithm fail.

The window size was replaced by target bit rate as we used the rate controller. This was used to determine the maximum bit in kbit by multiplying W with a thousand and making sure the bit rate was never under 300 kbps and never larger then 10000 kbps. This is due too as explained before how the rate controller works and that the standard W_{tresh} is set to 10.

6.3 TCP-CUBIC-FIT

TCP-CUBIC-FIT was first introduced in [36] where the authors combined the strengths from the old Linux standard congestion control TCP-CUBIC with TCP-FIT [37]. To understand TCP-CUBIC-FIT we need to first understand both of these congestion control algorithms.

6.3.1 TCP-CUBIC

TCP-CUBIC was the standard congestion control algorithm used in Linux between version 2.6.19 and 3.2 [16]. The algorithm is using a loss based approach and is designed to be friendly with other TCP congestion control algorithms [36] [52]. It is designed around a cubic function that when a loss occurs reduces the TCP window size and thereby also reducing the throughput of the service that runs the TCP connection. As time progresses and no further packet loss is detected the window size grows along the following function:

$$w_{cubic} = C(t - I)^3 + w_{max}, I = \sqrt[3]{(w_{max} \cdot b)/C} \tag{6.7}$$

In the function w_{cubic} is the new window size that is calculated every iteration. C and b are constants that decide how sharp the algorithm shall be when it modifies the window size. Finally t is the elapsed time since we last received a packet loss. This means that when a packet loss occurs and t is reset to 0, $w_{cubic} = -w_{max} \cdot b + w_{max} <=> w_{cubic} = w_{max}(1 - b)$. The functions behavior is that when t is small it grow rather fast and as it closes in to w_{max} it flattens out. The value w_{max} was the window size where we last got a package loss and therefore it is probably here that we have our network bottleneck. After we have passed w_{max} the function continuous to grow at an increasing speed. After a while the function will grow like a cubic function and hence will reach the maximum network speed faster as time goes on. It is important to keep the networks from congesting but it is also important that we don't under-utilize available bandwidth, that is the reason for the function to be more aggressive if we have been without a packet loss for a long time.

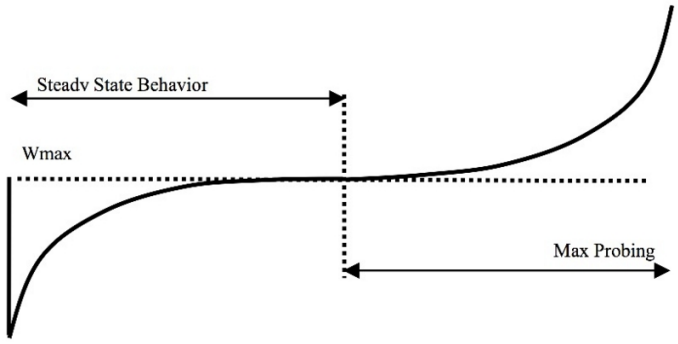

Figure 6.1: TCP-CUBIC growth curve after packet loss

6.3.2 TCP-FIT

TCP-FIT is used as an accelerator for a loss based algorithm and uses a multiplier N to increase or decrease the speed of the algorithm based on RTT [37].The value of N is thus either increased, decreased or stays the same every time there has been a packet loss. N depends on the average RTT window size $\bar{rtt}$ and the minimal RTT observed rtt_{min} were the latter is a estimate of the propagation delay of the network. N is updated according to the following:

$$N_i = \begin{cases} N_{i-1} + 1 & , Q < \alpha \cdot \frac{\bar{w}}{N_{i-1}} \\ N_{i-1} & , Q = \alpha \cdot \frac{\bar{w}}{N_{i-1}} \\ max1, N_{i-1} - 1 & , Q > \alpha \cdot \frac{\bar{w}}{N_{i-1}} \end{cases} \tag{6.8}$$

Where Q is:

$$Q = (\bar{rtt} - rtt_{min})\frac{\bar{w}}{\bar{rtt}} \tag{6.9}$$

Here $\bar{w}$ is the average window size of the i-th updating period. We can see that $\bar{w}$ is on both sides of the expression and can thus be expressed in a function where N_i is only dependent on $\bar{rtt}$, rtt_{min} and a scaling factor α. Theoretically α should be the quota between the number of packages that are queued into the buffer and the value of the congestion window. However these values are hard to measure correctly and an approximation is done where $\alpha = (rtt_{max} - rtt_{min})/rtt_{max}$. This approximation has been proven to bee good in according to both fairness to other TCP congestion control algorithms and throughput.

6.3.3 Combining TCP-CUBIC and TCP-FIT

TCP-CUBIC-FIT was first introduced in 2013 and is basically an implementation of TCP-CUBIC that uses the N variable from TCP-FIT to throttle its speed. The concept is easy, before a new congestion window is calculated the algorithm computes the value of N and then multiply N with the calculated congestion window size. If we have a low RTT the algorithm will reach full network utilization faster than it would have with only TCP-CUBIC. The algorithm then becomes:

$$w_{cubic} = N \cdot C(t - I)^3 + w_{max}, I = \sqrt[3]{(w_{max} \cdot b)/C} \tag{6.10}$$

Where N is defined like in TCP-FIT.

6.3.4 Implementation

In proper TCP-CUBIC-FIT we adjust the TCP window size when we get a congestion. In our implementation that would only affect a minority of the packages that we send since most of our data is transferred over UDP in RTP packets. The rate controller has been modified to enable the congestion control algorithms to change the target bit rate by just providing an integer. Since the CUBIC-FIT functions w_{cubic} and w_{max} values depends on each other and grow depending on the size of the other we can change them from being window sizes to be target bit rates instead. The algorithm will then find the maximum bit rate that do not causes packet loss. When packet loss occurs it reduce the bit rate by the

specified coefficient. As mentioned before the rate controller works best in the interval between 300 kbps and 50 000 kbps and this is not a thing that the algorithm takes into account. An modification had to be made to ensure that even if w_{cubic} drops bellow 300 kbps or pushes over 50 000 kbps the bit rate that is sent to the rate controller is never below 300 kbps or above 50 000 kbps. The final implementation does not exceed 20 000 kbps since higher bit rates was not needed and the maximum jump in bit rate is much lower compared to running on 50 000 kbps.

Chapter 7
Evaluation

We will now define quality of service, quality of experience and how we did our measurements. Finally we will present our result of said measurements.

7.1 QoS

Quality of service can mean many different thing for different people [25]. Thus we will start by defining our definition of QoS and then proceed to discuss its importance.

7.1.1 Definition

The service in our case is the transfer and receiving of data between two cameras. We define the quality as how well the transfer is done by looking at jitter, round trip time, packet loss, synchronization, target bit rate and actual bit rate the camera sends out. These will be measured and displayed in graphs.

7.1.2 QoS in video conferencing

The QoS is very important in video conferencing. RTT will have a great impact on the stream. If then the RTT starts to get high a queue may form which may result in dropped or delayed frames. Packet loss will also affect the quality as lots of packet loss may mean lots of lost frames. If the jitter is very large we also may get delayed and discarded frames as they might arrive to late. If some frames will take double the time to arrive compared to others they may simply be needed to be dropped as otherwise a delay will occur. The target bit rate of our algorithms will also have great impact and so will the actual bit rate. If either is to low we get under utilization of the system. A too high bit rate may mean that we get packet loss as we send out more than we actually can handle. Frame rate will also have an impact in video conference. If is very low the video stream will not be smooth.

As our algorithms is based on round trip time and packet loss these will have a even greater impact on the quality, as they will directly determine the bit rate which each camera will send with.

7.2 QoE

QoE differs from QoS in the way that QoS is the technical numbers on how good a service is and QoE is the measurement of how well the user of the service perceives it. QoE can have different technical properties and still have a high value. For example in a one system we might not be that interested in a high frame rate as we instead want the images to have high quality and the correct colors to be able to identify certain patterns in the video stream. This may however not be true when we are having a video conference. Here we may be able to accept a lower image quality as long as the frame rate is high and the audio is in synchronization with the video.

7.2.1 QoE in video conferencing

One important quality we found early, and one that had high priority by AXIS, was that the sound should not stutter. The reasoning behind this is that if the sound start to stutter, it can get very hard and/or frustrating to hear what an other person is saying. This is a major problem in a conference as hearing what the other part is saying and not get distracted by bad sound is very important. On the contrary the video might not be so important and may be even shut off completely as long as the sound works. While jests and such is very important for communication, or the need to show anything, it is not as important as audio in our case. Human are also more sensitive to audio distortions then distortions in video [53]. Thus it might be okay to sacrifice the quality of the video for a good audio quality. While we can sacrifice video to ensure audio, this may not always be the optimal solution. One could for example always have the lowest quality of the video displayed no matter what. This is not a good solution and thus we need to see how good quality we can actually get with the video without compromising the audio. This is to ensure the users the maximum quality of experience.

Another quality of experience to regard is delay. We found multiply sources that stated that the delay should be less or equal to 150 ms [24] [54]. Otherwise the users will start to notice. While you still may understand what is happening and what the other part is saying, it get frustrating and distracts from the call [24]. Thus is it very important to keep the delay to a minimum.

When having both sound and video the quality of synchronization is brought up. If audio or video is ahead of the other, it will also distract the users from the content of the call.

It is also important to look at the general quality of the video stream. Can you actually hear the other user and see him/her?

7.2.2 Measurment of QoE

According to Brunnström et al. [28] the best way of performing a QoE evaluation of an application is by using tests on real users of the application. QoE is a subjective entity and depends on unpredictable factors like users mood, expectation and context. To reduce the effects of the unpredictable factors test must be done on a sufficiently large group of end users with different backgrounds and expectations. This process is rather time consuming an not something we could afford to do. It would however provided the best result. The most common way to measure QoE is by subjective scoring [26].

There do exist objective methods/algorithms standards for testing both audio and video, such as ITU P.862 and ITU-T J.144 [27]. However they are quite complex and would again require to much time to implement.

7.3 Limiting bit rate

In the camera there exist a rate control already implemented. To evaluate the implementation we connected a computer and the camera to a Netgear GS110TP switch. This allowed us to limit the bit rate to a max of 1024 kbps. First we used no limitation at all to see what bit rate that was actually needed. This yielded us that when nothing happens or very little happens the camera used less than 1024 kbps.

However when movement occurred it needed to use more than 1024 kbps. Heavy movement came close to 7000 kbps. The rate control would not receive information about the limiting of the switch, resulting in heavy lag where only one out of many frames was shown. This indicates that when 3000 kbps needed, it will transfer frames with 3000 kbps, disregarding the actual available bandwidth. Thus there is a need for a network congestion algorithm.

7.4 Approach

To test the conference application we designed a few tests that will test different video properties on different types of network. All the tests are run on 720x576 resolution and audio is turned on. On all test we collect data from the stream and we focus on the data mentioned in our QoS definition.

The three different networks we used is a local network on Axis, our own home networks and a link to a camera in the united states.

- **AXIS:** The Axis network consist of two network cameras connected to a switch. The conference call are set up on the cameras via GStreamer launch commands. We then alter the switch's throughput by setting the maximum bit rate to 1 Mbps and then back to 100 Mbps. This is alternated in an interval of forty seconds on all tests with the exception of the one hour long test and video conference test. The video conference test will have an interval of one minute instead whiles the hour one will at all time run at 100 Mbps. The reasoning for altering the switch is to check if everything is working as it should in extreme changes of available bandwidth. Is the algorithm adapting? Do we still get audio and video? And so on.

- **Lund** The Lund network is the network between the two authors. Since the two authors both live in the same neighborhood the distance is small but the traffic from the video conference will compete with other data traffic since it's no longer a LAN.

- **USA** This link goes from the home of one author in Lund to an Axis office in the united states. The link is full of traffic and the distance is large. This test is mainly to see how delay impacts the streams and if the stream stays synchronized over long distances. The same tests are run at two different time periods, the first is during peak hours (between 20:00 and 0:00 local Boston time) and the second is done on low traffic hours (between 2:00 and 6:00 local Boston time).

On each network we want to verify that a few things work accordingly and thus the following tests was designed.

- The first test is a 3 minute test where the camera is faced to a wall and thus there is no motion on the picture. This test we call the no movement test and is supposed to be a baseline to see how the stream works with a very low load.

- The second test is called the motion test and it has also a duration of 3 minutes. Here we put the camera in front of a computer screen playing a video clip. The video clip chosen is a music video. This is because music videos are known to have a lot of motion in them. Can our application handle a music video it should be able to handle a video conference as well as generally a video conference does not have as much movement.

- The third test is called the alternating motion test and is the final 3 minute test. In this test we play a media clip which is totally still for 30 seconds and then has a lot of motion for 30 seconds. The clip is then replayed 3 times. This test is designed to test how well the application can handle big jumps in motion.

- The hour long test is the name of the fourth test. It is basically the third test run for an hour to check the stability of the application. It is also used to verify how the synchronization between video and audio fairs after an hour.

- The fifth test is the video conference test. In this test we use both cameras and connects an audio and video stream from each camera to the other. This test will verify if our application can handle a full conference call. The test is run for 5 minutes.

7.5 Resulting measurements

When we did our measurements we ended up with a lot of data. In this section we would like to show the most interesting graphs from tests that addresses the things that we will bring up in our discussion. Missing graphs will be available at the following website for the interested reader: `https://drive.google.com/folderview?id=0BzWy6HQ8so-Ja1hfV0ZWLXlPalE&usp=sharing`.

7.5.1 AXIS

In the Axis tests we note a change in the switch by a black square. The switch always start at full capacity. At the first black square it is throttled to it's minimum capacity and at the second it is released to its full capacity and it then alternates. A red circle marks when motion start and ends in the same way as the black square, starting with no motion and going to full motion at the first red circle.

Target bitrate

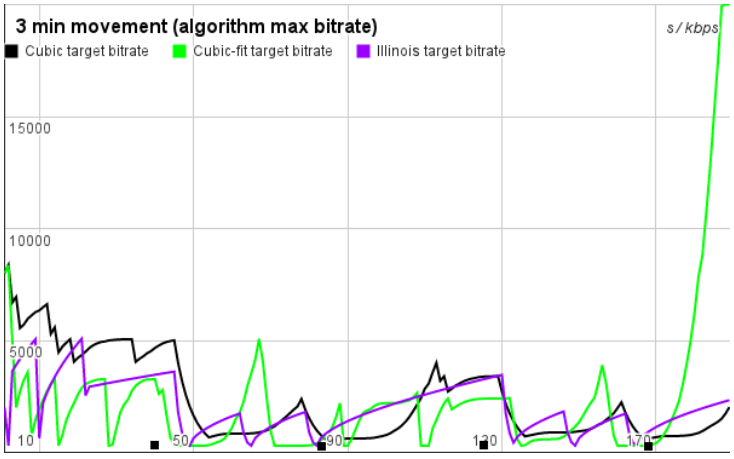

Figure 7.1: Target bit rate during the 3 min movement test at AXIS, We get a lot of packet loss here but we can see that the algorithms follow the expected behavior after wards

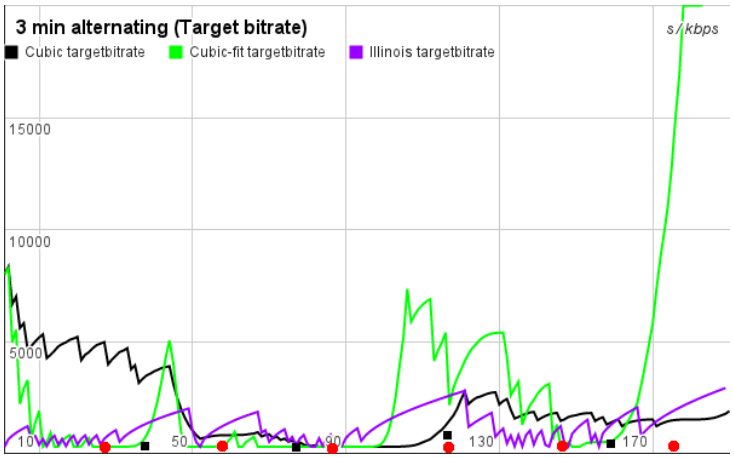

Figure 7.2: Target bit rate during the 3 min alternating test at AXIS.

Packet loss

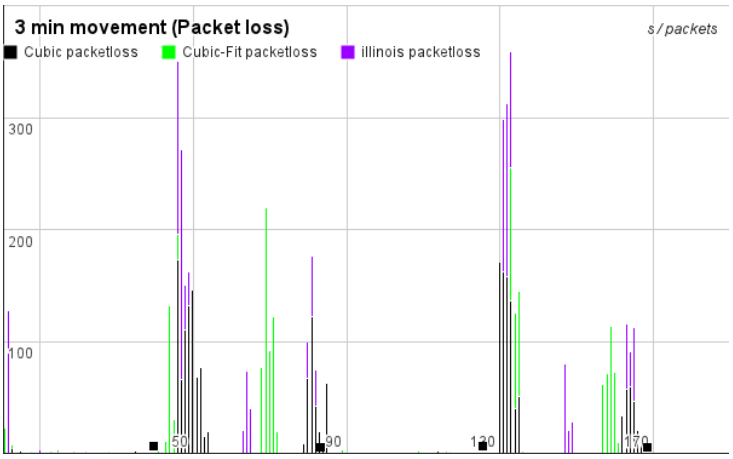

Figure 7.3: Movement packet loss. It is easy to see that the most of the packet loss occur directly after we limit the switch.

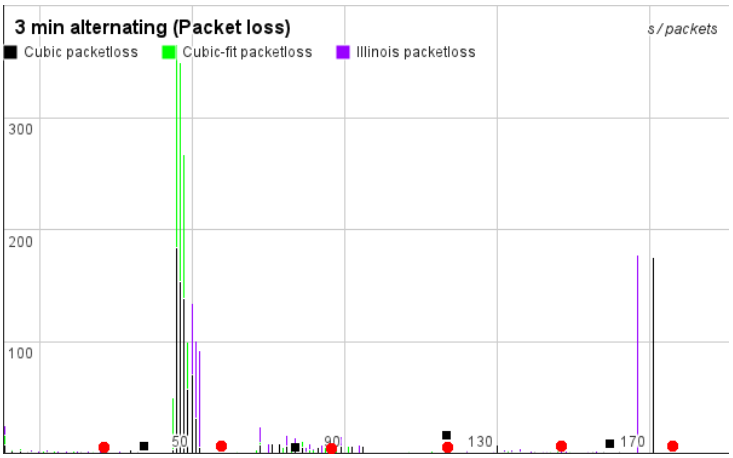

Figure 7.4: Alternating movement packet loss. Note that the we receive no packet loss when movement starts but after a few seconds with movement and limiting of the switch we got a lot of lost packets.

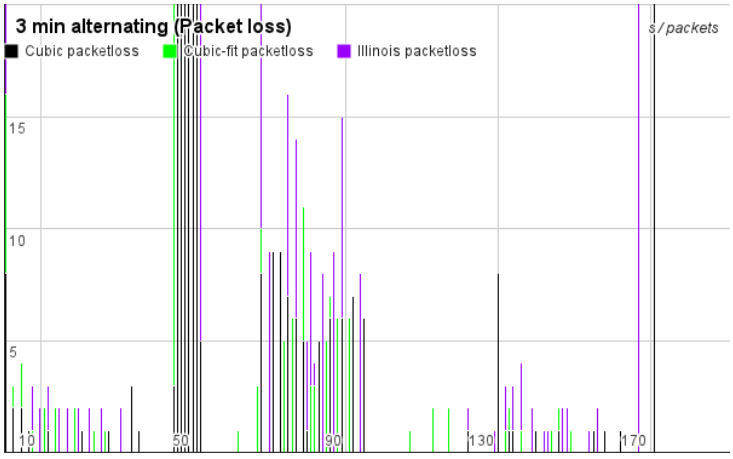

Figure 7.5: Zoomed in alternating packet loss. If we look close we can see that we get a lot of package loss when we have no movement at all, more about this in the discussion.

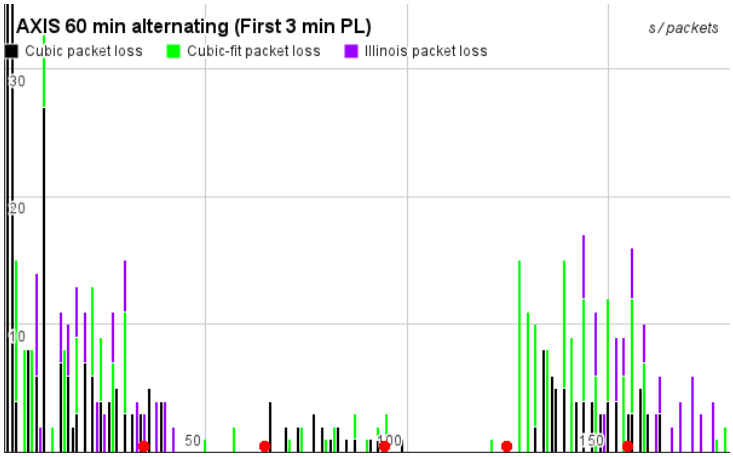

Figure 7.6: Packet loss of the first 3 minutes of the 60 minutes alternating test.

RTT

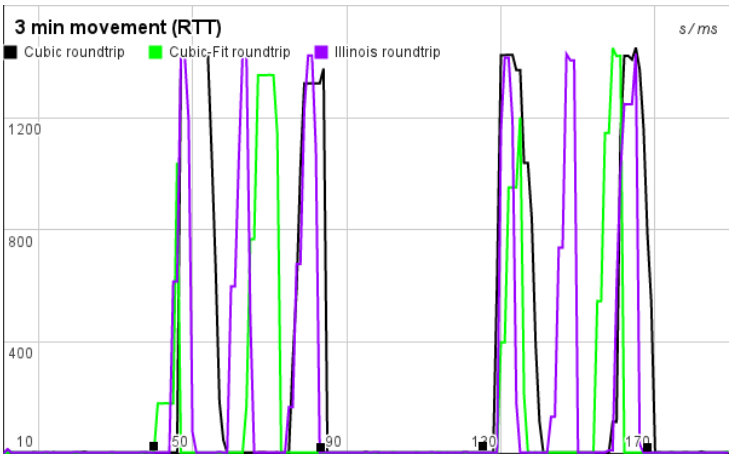

Figure 7.7: The round trip time increases a lot when we limit the switch wost likely strengthening our assumption that hybrid algorithms should perform better.

Jitter

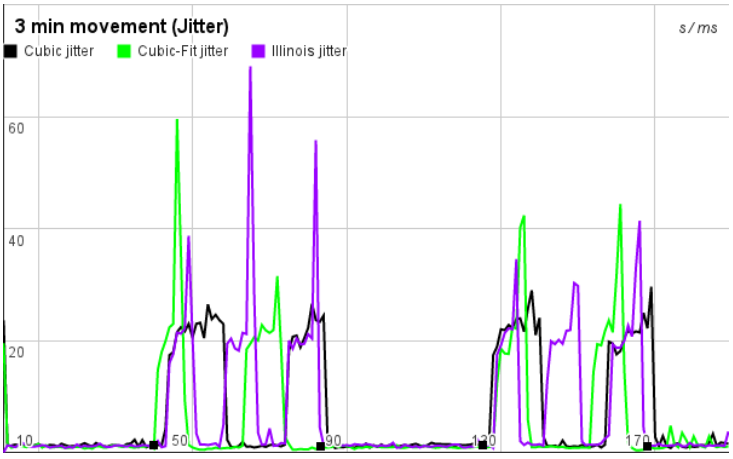

Figure 7.8: Jitter also increase when the switch is limited.

Actual bitrate

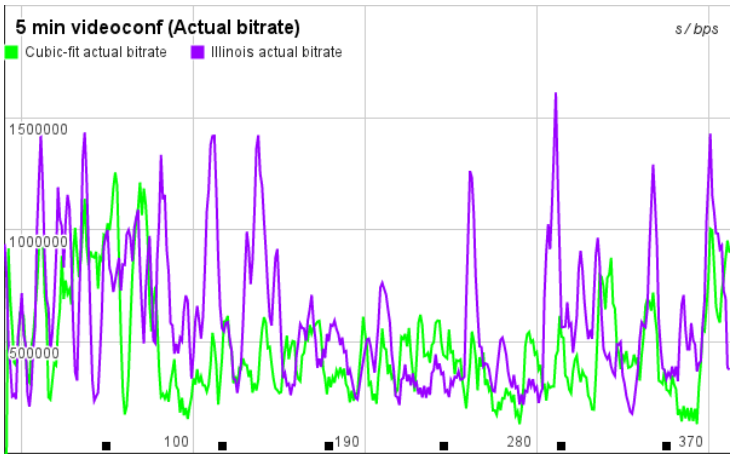

Figure 7.9: Here we can see that the actual bitrate is allowed to be lower than the target bitrate if the rate controller sees that there are little motion.

Frame rate

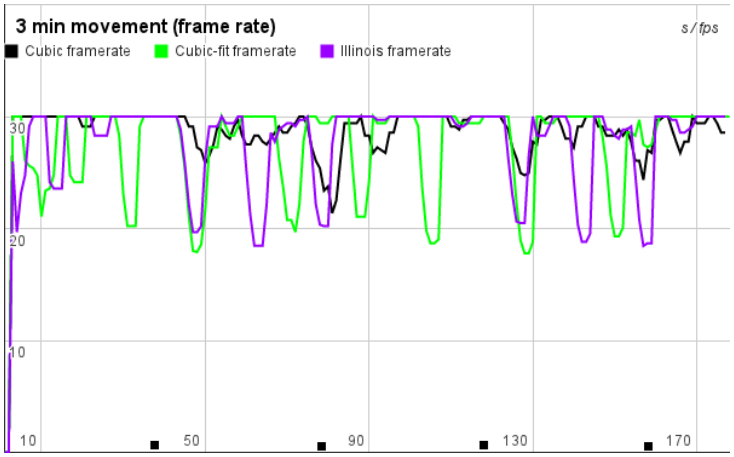

Figure 7.10: Frame rate of the 3 min movement test.

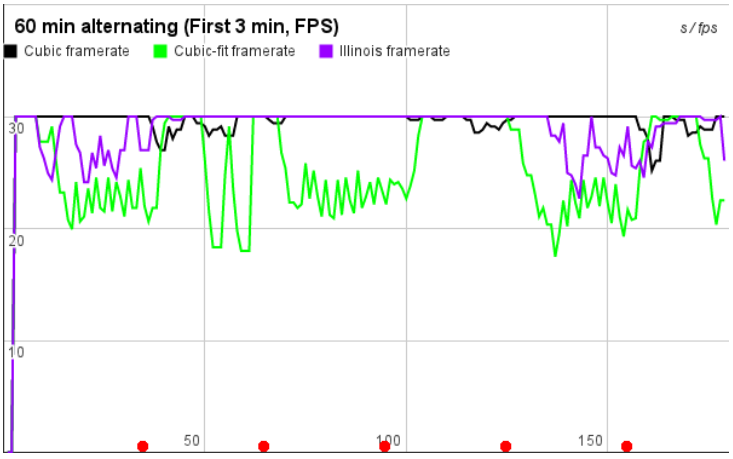

Figure 7.11: Frame rate for the first 3 minutes of the 60 min alternating test.

7.5.2 AXIS-PLL

Trying to improve the QoE we added a packet loss limit(PLL). If the packet loss was beneath a certain threshold, in this case 10, the algorithms would continue as if nothing happens. First when the packet loss was sufficiently large the algorithms would adapt. More about this in the discussion. The changes of movement and the switch are displayed the same way as previously.

Target bitrate

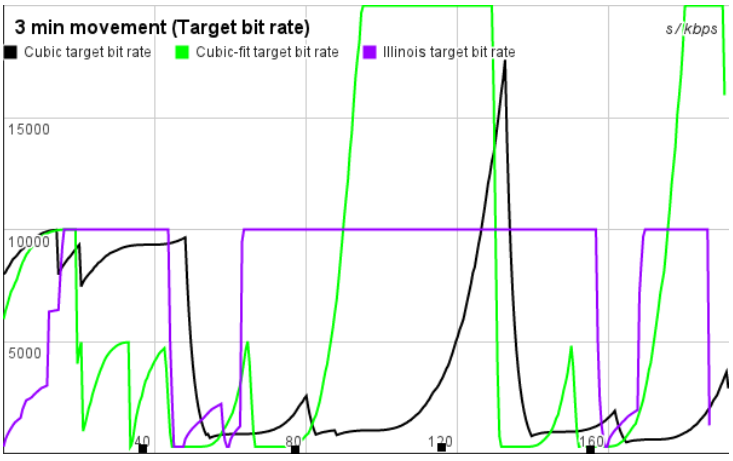

Figure 7.12: Target bitrate after introduction of PLL. The algorithms run smoother than without the PLL.

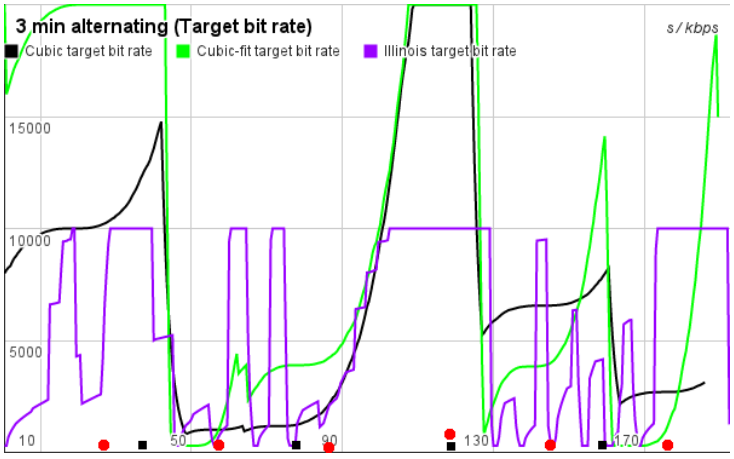

Figure 7.13: Target bit rate during the 3 min alternating test.

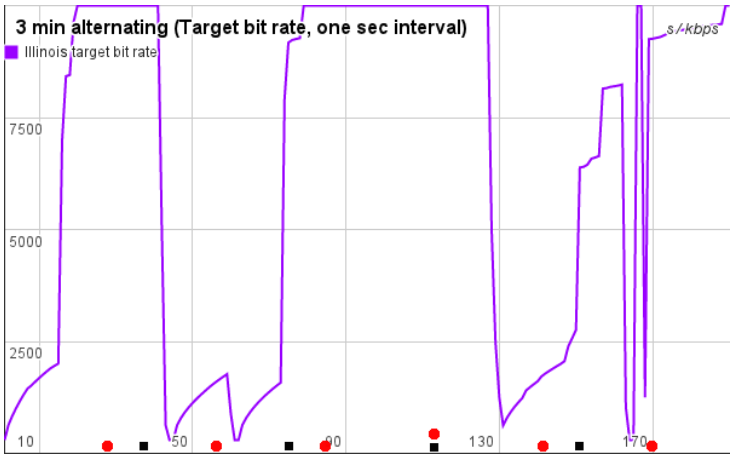

Figure 7.14: Target bit rate by TCP-Illinois during the 3 min alternating test. The interval the algorithm is running has been doubled to one second. We can see that Illinois is acting more stable than the previous graph.

7.5.3 Lund

Since the no motion test and the one hour test showed only little new compared to the other tests at AXIS and the ones to the USA, we only ran the motion, alternating and the video conference test between the apartments in Lund. In this test we could not limit the throughput of the switch but instead it should be the competing traffic that causes the packet loss. As in previous tests the first red dot represent a start in motion and the second a stop of motion and thereafter it alternates in said pattern.

Target bit rate

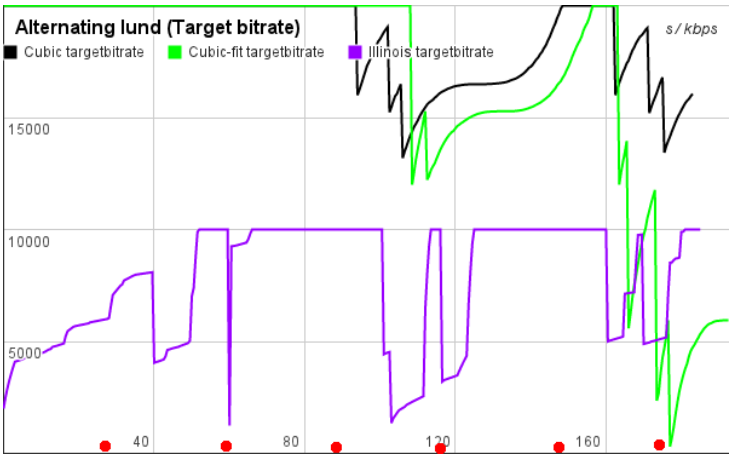

Figure 7.15: Target bit rate during the 3 min alternating test Lund.

Packet loss

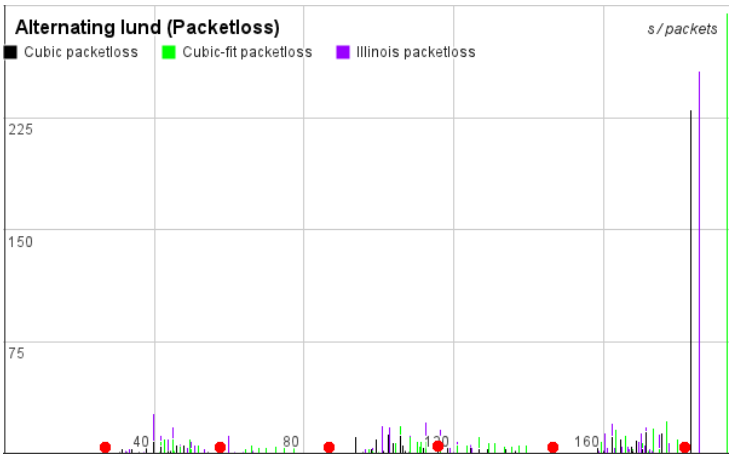

Figure 7.16: Packet loss during alternating movement in Lund. We can see that packet loss occur in conjunction with movement. The peaks of packet loss in the end of the graph is from when we shut down the stream during the tests(it expects that it should receive a lot of packages but since we interrupt the stream they never arrive).

RTT

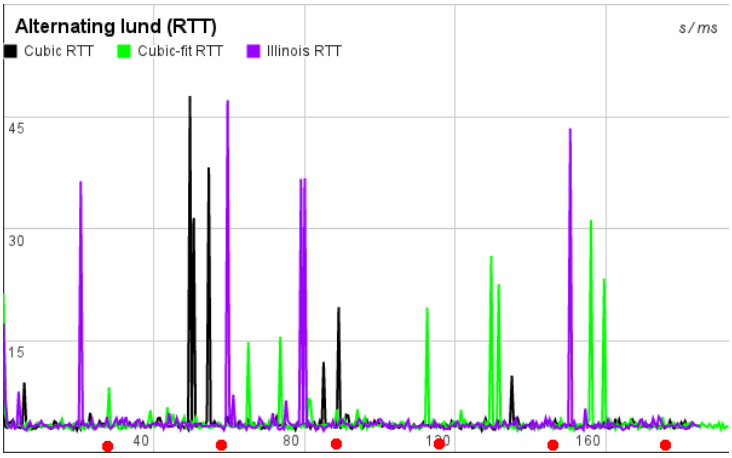

Figure 7.17: RTT during the 3 min alternating test Lund

Jitter

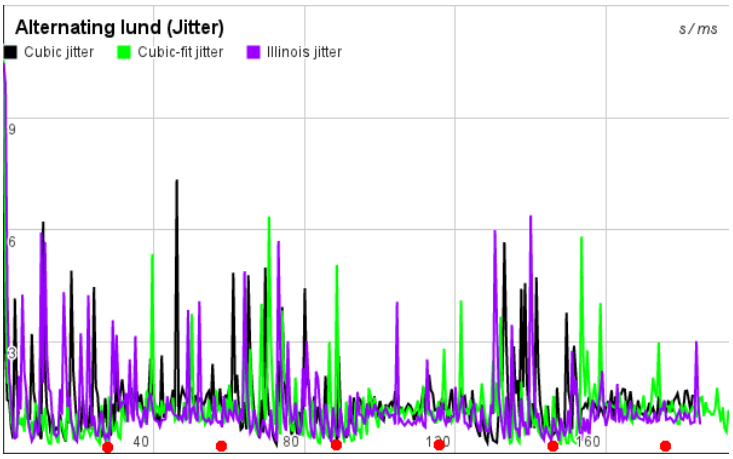

Figure 7.18: Jitter during the 3 min alternating test Lund

Actual bit rate

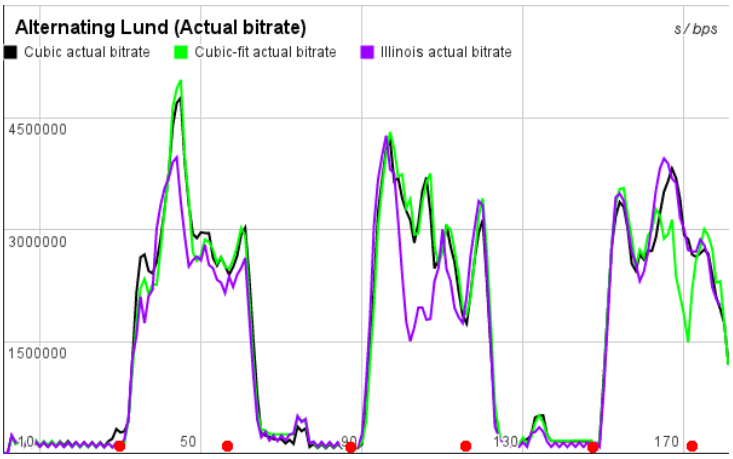

Figure 7.19: Actual bit rate sent from the server camera during the 3 min alternating test Lund

7.5.4 USA peak hours

The same tests as in Lund was run to the office in Boston. The tests were as mentioned run during peak traffic hours in USA.

Target bit rate

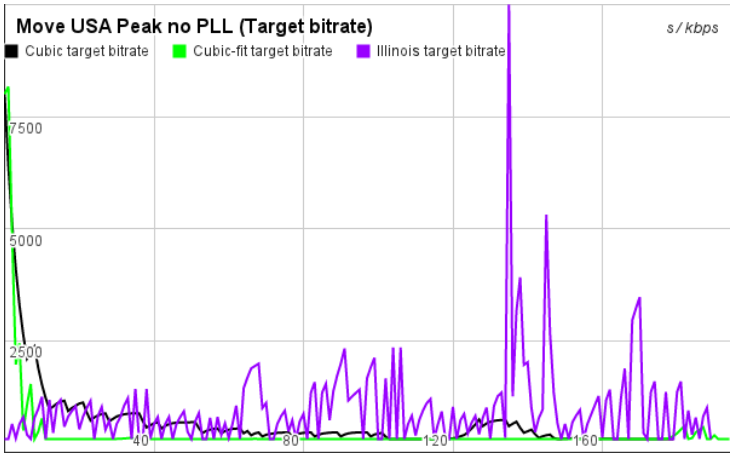

Figure 7.20: Target bit rate during the movement test to the USA

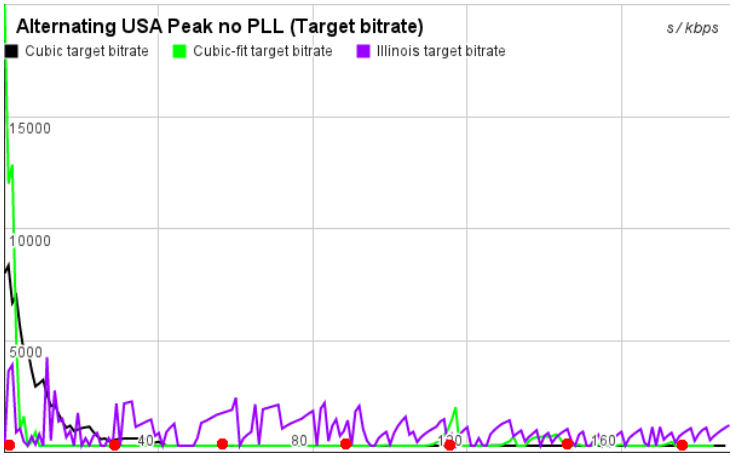

Figure 7.21: Target bit rate during the alternating test to the USA

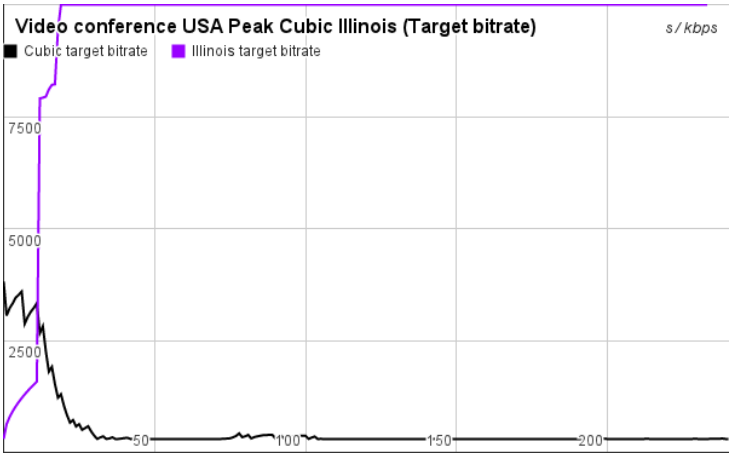

Figure 7.22: Target bit rate during the video conference test to the USA

Packet loss

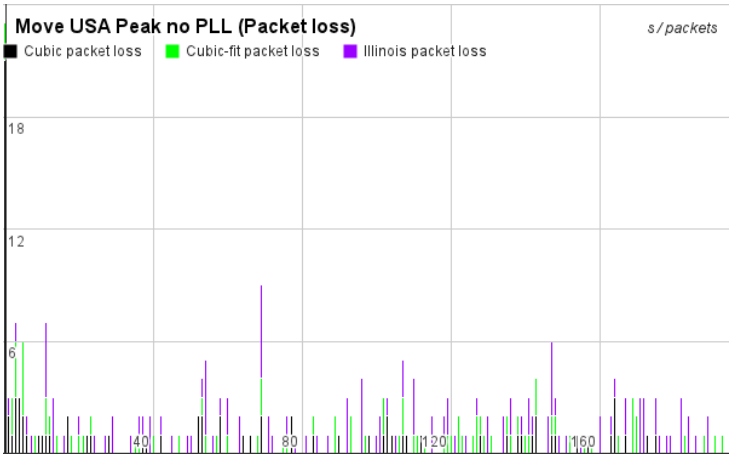

Figure 7.23: Packet loss during the movement test to USA. We can see very small amount of packet loss during the whole test.

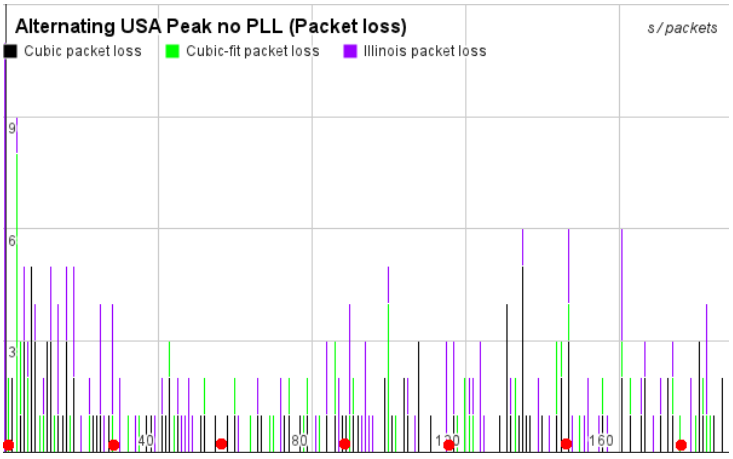

Figure 7.24: Packet loss during the alternating test to USA.

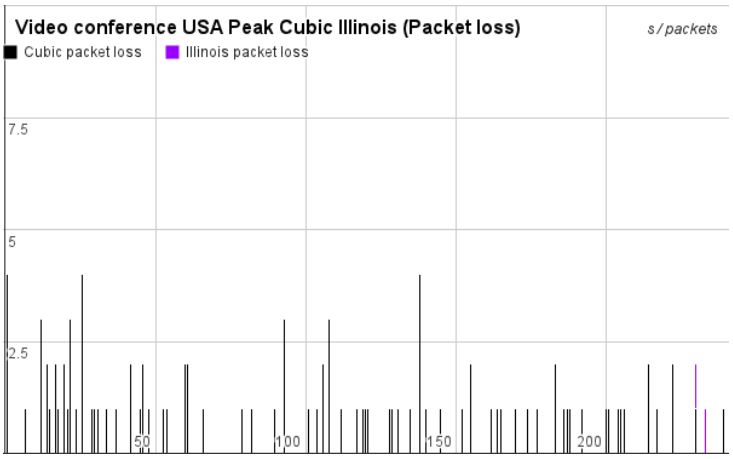

Figure 7.25: Packet loss over the video conference test to USA.

RTT

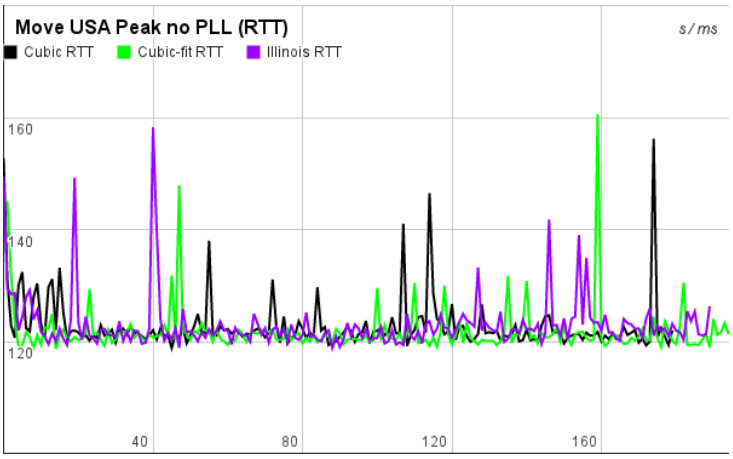

Figure 7.26: RTT during the movement test to USA

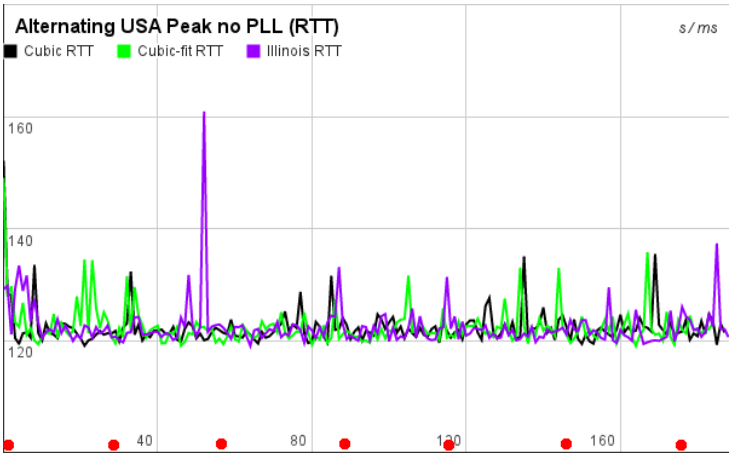

Figure 7.27: RTT during the alternating test to USA

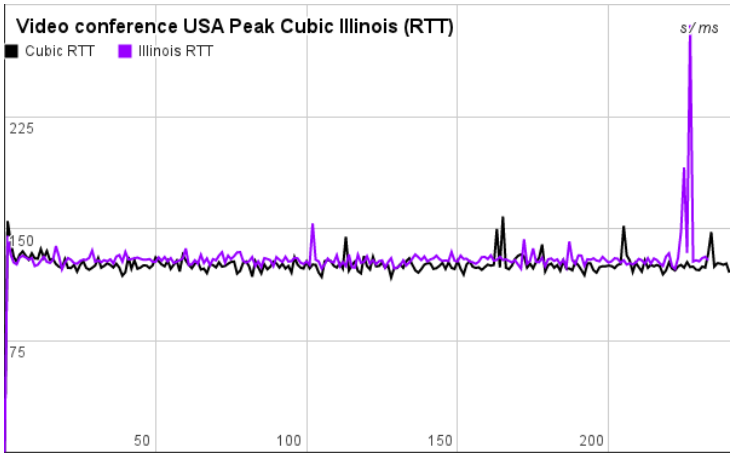

Figure 7.28: RTT during the video conference test to USA.

Jitter

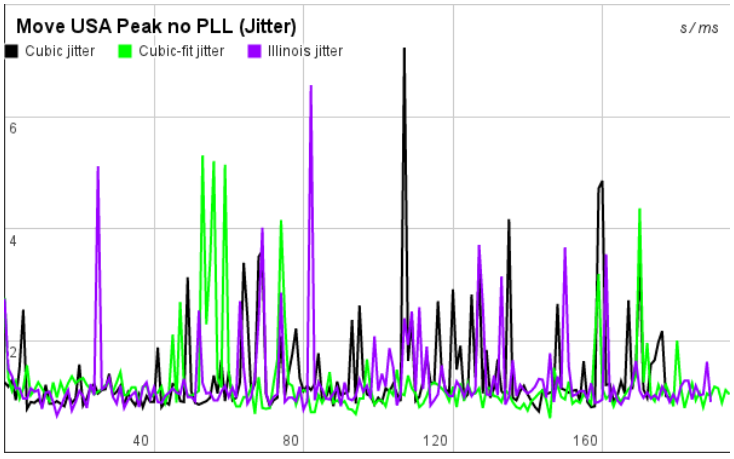

Figure 7.29: Jitter during the movement test to USA

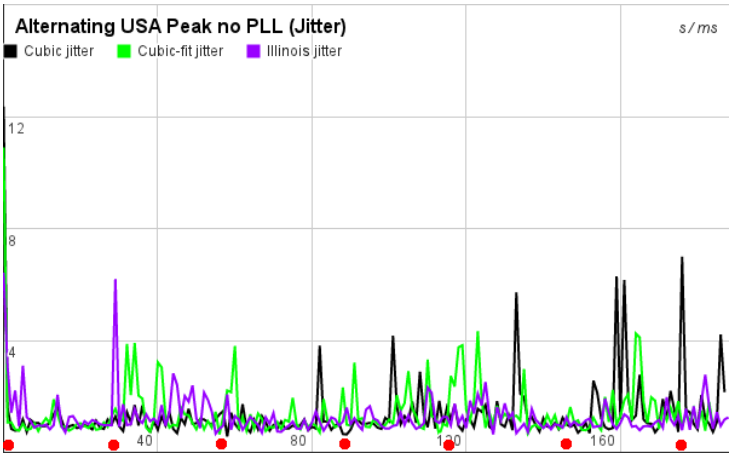

Figure 7.30: Jitter during the alternating test to USA

7.5.5 USA low hours

We chose to do only the alternating and movement test on the low hours test since those are the tests that has been most interesting. Video conference worked okay during high peak hours. We had already seen that synchronization was no problem with higher traffic so the 60 minutes test was deemed unnecessary as well.

Target bit rate

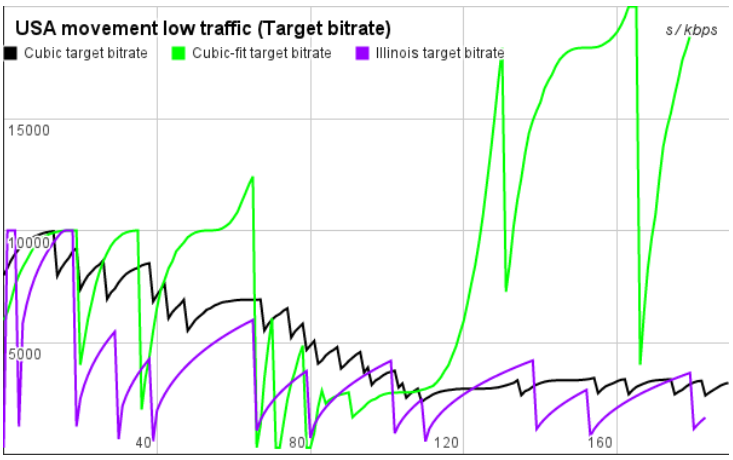

Figure 7.31: Target bit rate during the movement test to the USA

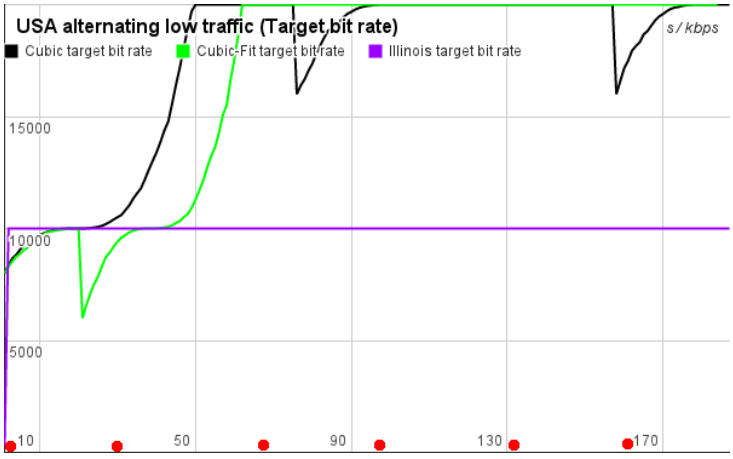

Figure 7.32: Target bit rate during the alternating test to the USA

Packet loss

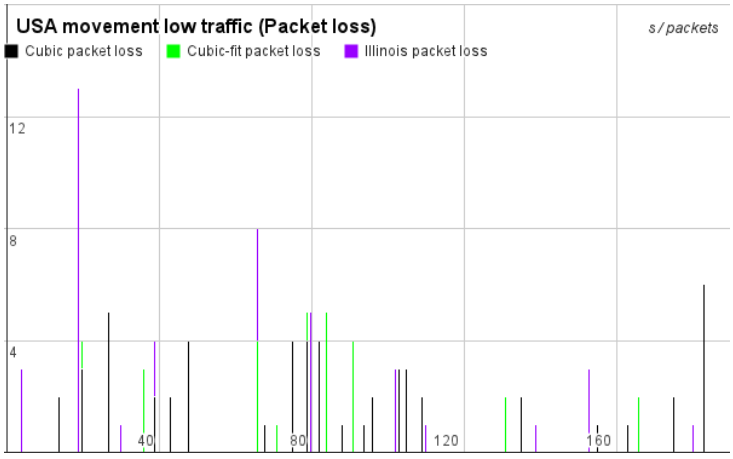

Figure 7.33: Packet loss during the movement test to USA. We can see that we got a lot less packet loss during low peak hours.

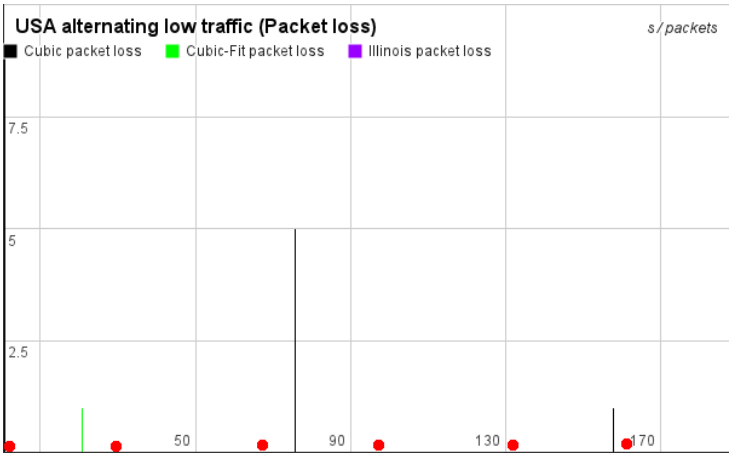

Figure 7.34: Packet loss during the alternating test to USA. Here we got even fewer packets lost than the movement test during low peak hours.

RTT

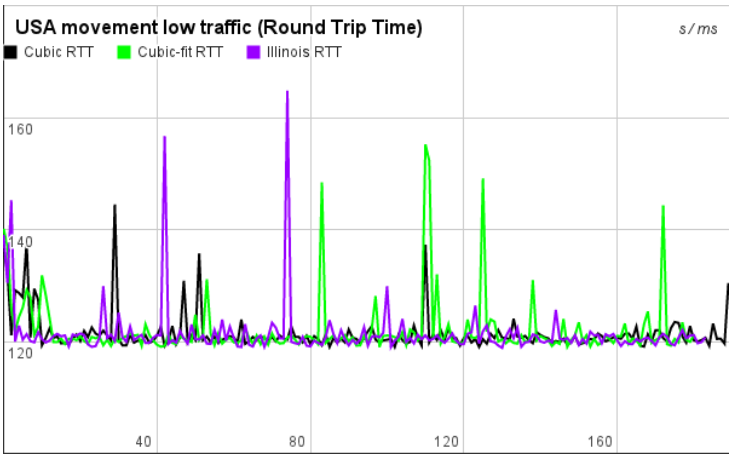

Figure 7.35: RTT during the movement test to USA

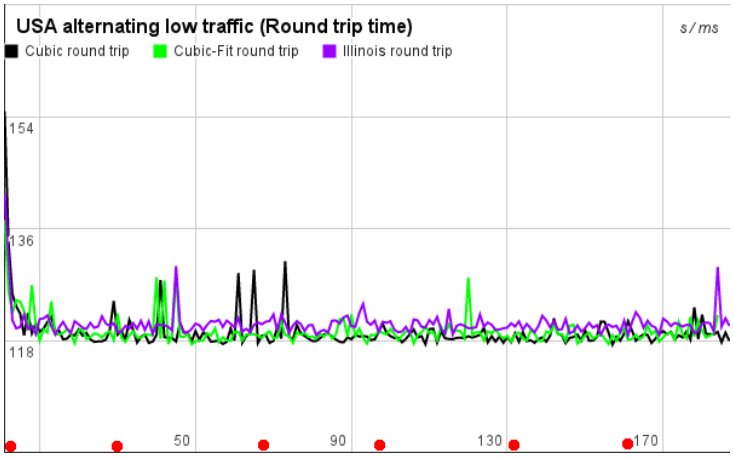

Figure 7.36: RTT during the alternating test to USA

Jitter

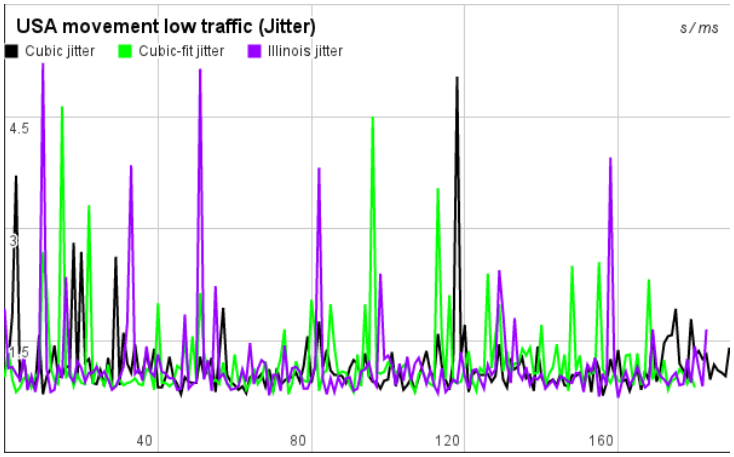

Figure 7.37: Jitter during the movement test to USA

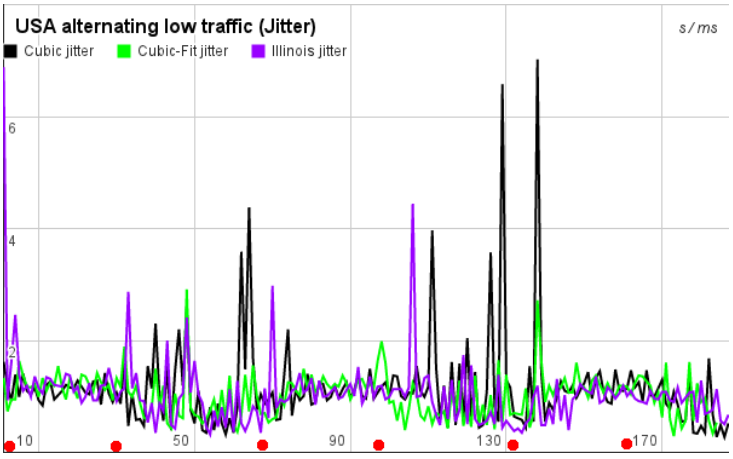

Figure 7.38: Jitter during the alternating test to USA

7.6 Packet loss

A common way of measuring packet loss is by percent. We have here summarized how much packet loss we got.

7.6.1 AXIS

At AXIS we reduced the bandwidth a lot on the switch for almost half of the playback time. Therefore the packet loss on these tests are very high.

3 min alternating

Cubic	5.5%
Cubic-fit	3.9%
Illinois	3.1%

3 min movement

Cubic	8.1%
Cubic-fit	6.2%
Illinois	5.7%

60 min alternating

Illinois ran a version of the algorithm that updates twice as fast as the previous two tests.

Cubic	1.1%
Cubic-fit	0.97%
Illinois	0.34%

7.6.2 AXIS-PLL

As mention we added a PLL to try to improve the QoE. This is the resulting packet loss rate. On almost all accounts the packet loss increased, but when we did our QoE tests the video clips with the PLL implemented got a higher score.

3 min alternating

Cubic	11.6%
Cubic-Fit	8.1%
Illinois	10.8%
Illinois one sek interval	7.4%

3 min movement

Cubic	13.6%
Cubic-fit	6.3%
Illinois	5.3%

7.6.3 Lund

In the tests in Lund between the two apartments we got no or close to no packet loss at all on both the video conference test and the constant movement test. Thus we did not show the percentages on those. The only time we got significant packet loss was on the alternating test.

Lund alternating - PLL

Cubic	0.80%
Cubic-fit	1.0%
Illinois 0.5 interval	0.95%

7.6.4 USA peak hours

During peak hours the strain on the network is much higher than on the low peak hours and thus more packet loss occur. However all our tests had a packet loss of around one percent, which was the threshold for video conferencing, and therefor passes our packet loss requirement.

3 min alternating

Cubic	1.1%
Cubic-fit	0.82%
Illinois	1.1%

3 min movement

Cubic	0.80%
Cubic-fit	1.1%
Illinois	0.97%

5 min video conference

Cubic	1.1%
Illinois	0.61%

The vast majority of the packet loss Illinois experienced happened during the first second. Ignoring these yields us:

Illinois	0.01%

7.6.5 USA low peak hours

Since the video conference test had shown adequate performance during high traffic hours, this test was deemed unnecessary during the low peak hours.

3 min alternating

Cubic	0.06%
Cubic-fit	0.038%
Illinois	0.012%

3 min movement

Cubic	0.20%
Cubic-fit	0.16%
Illinois	0.19%

7.7 Hardware Performance

To make sure that our algorithms did not take up to much processor power and memory we made measurements during our tests. We first measured how much memory and cpu the camera used when connecting one audio and one video stream to another camera. We measured both the server and client side to see how much performance the different roles took respectively. The next performance measurement was conducted on two cameras running streams to each other to see how demanding it is for the camera to both receive and send a live stream. Finally we ran the stream for an hour to make sure that everything was stable.

7.8 Evaluation process

The evaluation done from the perspective of QoS was done by simply looking at the statistics we had available. We could see as an example how much packet loss we had and where, letting us draw important conclusions.

From a QoE perspective we simply watched the videos we recorded and analyzed them. We both have used Skype before as a video conference tool and based on our experiences we could conclude what we though was okay or not. In order to more easily compare the result we also started all the videos from the same test at the same time to better see the differences.

Chapter 8
Discussion

Here we will discuss our results and our findings. We will also reflect on the results we got from our QoE evaluation. The video conferencing implementation was not as easy as we first thought, there were a lot of parameters that we assumed worked as intended but in fact did not which has affected the results in different ways. These will be addressed as well.

8.1 Local network tests

The first test was the no movement test. The objective of this test was to let us confirm that the algorithm was running and that we could get the statistics we wanted. As this worked fine we quickly moved on to the other tests.

What we noticed quite quickly when looking at the statistics and the videos from all local tests, except no movement and video conference, was that we quite often got seemingly random packet losses. The network in it's self should be perfect and we expected no packet loss at all unless we limited the switch. There was no competing traffic on the switch and it was set at 100 mbps. This can be observed in 7.5. As the algorithms is supposed to decrease the QP value of the images by reducing the target bit rate when receiving packet loss, the target bit rate therefore sort of a fluctuating pattern. The target bit rate would go down simply to then rise quickly only to go down again. This behavior decreased the QoE of the videos which we will discuss later. Unfortunately we never discovered why the packet loss occurred when the network was at full speed. Connecting both cameras to a computer with two network cards and routing the traffic through removed this problem. Changing switch to an other type did not work. Running the stream to a computer through the switch instead of camera to camera worked flawlessly. In the end we needed to move on due to time constraints but our theory is that it might have something to do with how the cameras communicate an sets up a connection with the switch. Since the cameras can stream video to a computer without any problems we think that the problem might be sit-

uated at the communication through the switch. Another theory is that it is the switch in itself that we used since the packet loss nearly disappeared when we did the tests in Lund.

In order to work around this issue we simply put a limit that 10 packet losses needed to occur before a change of target bit rate was performed. The limit of 10 packets was concluded by looking at the statistics. These random packet losses was almost never larger than 10 packets at a time. This limit had the effect of greatly decreasing the fluctuating patterns and improved the QoE according to the authors. Two other master thesis workers was also asked to see if they came to the same conclusion. The video was displayed to them without explaining what the difference was, just asking them which one they though had better quality. Both came to the same conclusion as the authors.

Interesting to note though is that these random packet loss reassemble the characteristics of a wireless network. In wireless networks random packet loss may occur, but hopefully not as often as we experienced.

8.1.1 Movement

During the movement test when comparing the algorithms reaction time we could see that Cubic-Fit and Illinois most of the time performed better than Cubic. What we could observe was that when packet loss occurred due to reducing the bandwidth of the switch and combining that with high movement, both Cubic-Fit and Illinois reduced the target bit rate faster than Cubic. They take in account the RTT, and as seen in 7.7, the RTT increases a lot when a limitation of the switch is done. An increase in RTT is often a clear signal that a congestion is imminent and soon to happen. This causes Cubic-Fit and Illinois to respond aggressively. Cubic however do not look at the round trip time and simply see that packet loss occurs. It will try to slowly decrease its target bit rate to make sure it does not decrease to fast. In our case we heavily decreases the bandwidth available on the network which in turn more negatively impact Cubic in regards to its reaction time. This can be observed in graph 7.1 at around the 50 or 130 second marks. Note that the time it takes for Cubic to react compared to the others is several seconds.

A solution where instead of prioritizing packet loss as the trigger mechanism for the algorithms, we could use a delay based algorithm where we look at the RTT as the first parameter. Such a solution might have a better reaction time since we can see in the graphs 7.7 and 7.3 the RTT increases before we get packet loss when limiting the switch. We can with other words react even before we have gotten packet loss. But as discussed before, such as solution have problems too. In our case we can even add a large problem, namely the RTT threshold. Since a call from another continent has a much higher RTT than a local call this threshold would have to be manually changed depending on the distance between the two cameras. Alternatively use, as in our implementation, a ratio of the average RTT and the current RTT.

If the round RTT is low we probably still can send with the previous bandwidth as there probably is no congestion. Thus the algorithms tell the rate controller to rise quickly again. This is generally a good thing as it lets us fully utilize the available bandwidth. This is something both Illinois and Cubic-fit does but not something regular Cubic do. This can be observed in 7.1 around the 10 second mark, where the RTT is low. Here both Cubic-Fit and Illinois increases its target bit rate faster than Cubic will ever do.

In regards to the packet loss rate, it is clear that both Illinois and Cubic-Fit are much

better than Cubic. In cases the packet loss rate is more than half than that of Cubic. From a QoS perspective it is clear that Illinois and Cubic-Fit outperforms Cubic. As [10] mention, 1% loss rate is a recommended maximum amount of loss a video conference shall experience. None of the algorithm passes this. However in this test there is an unusually amount of movement. We also heavily decreases the available bandwidth to really put the strain on the algorithms and the random packet loss is still there. All these factors will cause extra packet loss, in a situations that's perhaps not likely in a real video conference, making this acceptable according to the authors. It is at the critical points, such as limiting the switch with high movement, where this is noticeable and we get heavy lag. This can observed in 7.3. Otherwise such as when the switch has been limited for a while and we still have movement the quality appears to be okay as the algorithms have had time to adapt and we get fewer packet loss. Note that we still get packet loss in the middle of movement intervals from Illinois and Cubic-Fit. This is because these will probe with higher and higher bit rate faster than cubic due to low round trip time. As mentioned, we have high round trip when we have packet loss, otherwise we have a low round trip. Cubic on the other hand probes more slowly and only get packet loss at the end of the movement interval.

Looking at the frame rate the server camera is sending out Cubic generally at a higher frame rate. This is seen in all test and is not exclusive to the movement test. It is however easily seen here 7.10. This suggest that Cubic would be better as it has higher average FPS, however this is a bit misleading and is not actually the case. The rate control will lower the FPS if it sees that it can not withhold the current target bit rate with only changing the quality of the frames. As Cubic react more slowly with changing its bit rate it will send with higher FPS, but this is actually more than what the current network can handle, causing more packets to be lost. This results in Cubic having the lowest frame rate when looking at the stream on the client side. Both Illinois and Cubic-Fit will send with a lower frame rate, which is more adapted to the current network conditions and thus all or most of the frames will be received at the client side resulting in a better QoE.

The differences between Illinois and Cubic-Fit was hard to see from a QoE perspective. The videos with the algorithm looked more or less the same quality wise. It is hard to draw any other conclusion than that they perform about the same by simply looking at the videos one at a time. Playing all at once made it clear that Illinois and Cubic-fit performed a bit better than Cubic, but not by as much as we though. We could see more lag with Cubic but only because we knew what to look for. Cubic is clearly not as good, but we are not sure if an ordinary user actually would be able to notice the difference.

8.1.2 Alternating

In the alternating test we discovered an interesting thing, the stream would freeze during the parts of the clips where there were no motion and run smoothly on the parts where there were a lot of motion. Graph 7.2 show that the target bit rate fluctuate a lot and stays low for Cubic-Fit and Illinois during non movement.

One would think that the likelihood of a packet loss would be higher when we have a higher bit rate caused by a lot of movement since it put more strain on the network than when we have a low bit rate. The author of the rate controller was contacted and the issue was discussed. He though it might be because of a setting in the rate controller.

The rate controller used the previous frames as input to calculate if the next frame

should be dropped to ensure the timings. As the quality goes from very high to very low the rate controller can see that the last few frames consists of a lot of data and when it looks at the current rate, it sees that there is no more room for frames with the currently set target bit rate and thus must throw the frame. In a surveillance system it might be important that the frames are not lost and therefore we have this setting. When there is no guarantee that all the images will be received at the client it is better to drop a few images to make sure that at least some images are received. There is no problem to experience a few moments of lag, the important thing in this scenario is to be able to identify a perpetrator and this is possible even if the video is not fluent. However in our case you would prefer a best effort approach where if a packet is lost we just carry on.

When the movement started and the switch network speed was reduced there were a period of lag where the algorithms adjusted the bit rate accordingly to the new network conditions. This delay was 2-3 seconds shorter on Cubic-fit and Illinois than regular Cubic. Cubic-fit was about 1 second faster than Illinois. The Illinois algorithms parameters where tuned to have a shorter update interval resulting in Illinois claiming the fastest adjustment time of 5 seconds, shortly followed by Cubic-fit at 6 seconds and finally Cubic at 9. Here we could clearly see that both Cubic-Fit and Illinois outperformed Cubic, reacting a lot faster.

As mentioned in a previous chapter we introduced a packet loss limit to counteract the lag we had in the part where there were no motion. The PLL did actually reduced the lag substantially on these parts of the recording hinting that it is a combination of the settings in the rate controller and the random packet loss that creates the lag phenomenon.

An interesting note is that we actually got a higher packet loss rate with introduction of the packet loss limit in this test. This is most likely due to that we do not adjust the bit rate for every time we get a packet loss, increasing the bit rate we send with. Without the limit we will instead adjust the target bit rate for every packet loss, causing it to go down more often and thus the camera will send with a lower bit rate. This will cause smaller packets to be sent, increasing the chance for them to not be lost. This would imply that using the limit is actually worse. But from a QoE perspective the videos got a lot better with the PLL, even though we got more packet loss. As mention in [2] a constant bit rate is better than constantly changing it. As explained we got the fluctuating pattern when not having the packet loss limit. This goes against what we just said, to keep a constant bit rate. By constantly changing the bit rate it gets confusing for a user as the quality keeps changing. In our case the high rate of changes caused freezes, even when there was little movement. The algorithm does what they are supposed to do, optimize the use of the bandwidth, causing less packet loss. But as seen this is not always the best cause for the user. An important lesson to learn is that even though QoS may say that a specific algorithm is best, this may not be true from a QoE perspective. This also highlight the importance of testing from a QoE perspective. In this case the video clips with the PLL fix and the one without the fix both have lag in them so the conclusion that one can make, is that if we have a lagging stream, we can accept more freezes if the quality of the images we get is better than the opposite.

There did however exist points from a QoE perspective that was better without the packet loss limit. When we limited the switch the use of the packet loss limit would cause the video to freeze for about 2-3 seconds longer than without. As the bit rate was generally higher it needed to go from a higher quality than without the packet loss limit to a lower.

This took a little bit extra time.

Looking at the statistics when using the packet loss limit we can see that Illinois have more of a squared peaks pattern than both Cubic and Cubic-Fit regarding the target bit rate, seen in 7.13. This is generally not good as for the same reason as the fluctuation pattern. The reason for that behavior is the parameter theta in Illinois which currently is set to 7. This roughly means, as the algorithm ran about every 0.5 seconds, that if the RTT is okay for around 3.5 seconds it is okay to go to the maximum target bit rate. But at maximum it will soon experience a new packet loss, going back down. These around 3.5 second intervals can easily be seen in the graph where Illinois slowly pokes and then suddenly uses its maximum target bit rate. Increasing this interval will increase the time it probes and the time we need to have good RTT before going to the maximum allowed target bit rate. This will likely lead to better result in these situations. On the flip side cases where we want to go to the maximum fast we will instead probe for longer. This may cause under utilization of the bandwidth. This parameter is with other words quite important and may need to optimized depending on the situation at hand. Due to this fluctuation Illinois have a lot more jitter and packet loss than Cubic-Fit. Cubic-Fit is in this case a clear number one. From a QoS perspective it seams that CUBIC ones again will perform worse with Cubic-Fit taking the top spot. It is worth noting that the difference was not as large as the movement test. CUBIC had for example no longer more than a double loss rate compared to the other two.

A test was done to see what implication would be of changing how often Illinois was running. We slowed down the interval the algorithm ran to once every seconds, effectively having the same implication as changing the theta to the double. This instead gave a packet loss rate of 7.4%, beating Cubic-Fit and making it overall perform better. The target bit rate is now slower, as can be seen in graph 7.14. As we can see, the parameters can have a drastic change on the performance of an algorithm.

Generally we could see that when we had movement it works good, as with the movement test. With alternating however it works worse at the switches in movement. The tests sudden shifts in bit rate causes problems for the rate controller. This can also be noticed, as explain, with the use of the PLL.

8.1.3 One Hour

In this test we did not limit the switch since we had already done this in the alternating three minutes test. As seen, this gave us a lot lower packet loss rates. Not limiting the switch caused more packets to be able to arrive safely.

It is important to note that the algorithm reacted to the random packet losses during this test as this was before the packet loss limit was introduced. This is because the 60 min tests was done primarily to see if we still had synchronization after a long period of time. We also wanted to see if it was possible to run the algorithms for at least an hour to make sure that we had a stable system.

As before we could note that we had freezes during no movement and no freezes when we had movement. It was especially visible due to not changing the switch. As we do not do this, a closer look at the packet loss graph 7.6 show this phenomenon clearly. Packet loss still occurs though as seen, still causing lag as we have a fluctuation pattern. These random packet losses did however actually served this purpose well as we could still see

the algorithm adapting to them even after 60 minutes, which was one concern of this test. An interesting note is that during the alternating test two clips was repeated. This can also be observed as one had clearly more movement, causing more packet loss. Note that Illinois is a couple of seconds delayed in the graph. It is very important to mention that Illinois do not get packet loss during the clip with lower movement. This was due to the computer screens angle being slightly different. The test was not repeated as it did not affect the main concern of the 60 minutes test.

The FPS is reduced in these interval of packet loss for Cubic-Fit and Illinois, seen in graph 7.11. The rate controller will try to adjust the FPS to better accommodate the target bit rate. Cubic-fit does not appear to have the same drop, nor Illinois. Illinois reason is as explained, due to the change of angle. Cubic as before does not react as fast as the other, causing it to more or less never go down to its minimum. The movement in the frames can be handled by the bit rate Cubic uses, but not for example when Cubic-Fit uses its minimum, so it need to lowers its FPS.

More importantly we could conclude that GStreamer even after 60 minutes was able to handle synchronization just fine. There was no issue there at all. The packet loss rates for both Cubic-Fit and Illinois passes the 1% requirement, but cubic does not. As we had a low RTT when we had packet loss, we did not limit the switch, both Cubic-Fit and Illinois was able to react faster then Cubic, causing less losses.

From a QoE perspective we conclude that Illinois was a lot better than Cubic and Cubic-Fit. This comes as no surprise as Illinois had less packet loss and a lot higher target bit rate due to the angle, subsequently also causing the camera to send out with a higher bit rate. More importantly the client was able to receive all packets even with the high bit rate. Between Cubic and Cubic-Fit there was not to much difference, the experience was pretty much the same. We could however conclude that in the switches from movement and no movement, Cubic-Fit tended to react a bit faster, causing less lag in these switches. The difference is however small, around 1-2 seconds. The overall experience will probably not be affected to much from this difference if there's not a lot of switches. In that case Cubic-Fit is probably the better algorithm of the two.

We did redo, not the whole test, but a part of it to see what implication the packet loss limit would have. With this all algorithms pretty much held the max target bit rate all the time. The user experience was a lot better for all the algorithms. Instead of low quality due to the lower target bit rate and lags that happened as we constantly changed the target bit rate due to packet loss, we got a smooth video with higher quality and a lot less lag.

8.1.4 Video conference

During the video conference test we pretty much had no packet loss at all and the videos from a QoE perspective was very good. In fact it looked a lot like the no movement test in terms of quality. What we noticed was that the actual bit rate the camera was sending out was almost always lower than the lowest bit rate we could set on the switch, seen in 7.9. This meant that even when we limited the switch there was no packet loss. Apparently the movement in a video conference is sufficiently low to not affect the bit rate as much. Running H.264 in conjunction with low movement reduces the bit rate a lot. The P-frames contains nearly no data at all and therefore the only data that needs to be transmitted is basically an I-frame once in a while. As we can see in graph 7.9 the bit rate on some places

were almost as low as the bit rate for the sound which we previously stated already was quite low. During this test we no longer experienced random packet loss. We are uncertain as to why this issue no longer occurred. One reason could be that the rate controller settings doesn't have any impact here anymore because we always have sufficient movement to not cause it to appear. As seen in the alternating and 60 min test, when movement occurs these settings causes no issue. Another factor could be that the movement is not very high, and as mention it causes a bit rate that is lower than the limit of the switch. Perhaps the sufficiently high movement and the sufficiently low bit rate in combination may explain why these random packet losses have disappeared.

8.1.5 Jitter

Looking at jitter in all the local tests we could see that when jitter occurred, due to for example limiting of the switch, Cubic had a longer period of jitter than the other two algorithms. However the other two had higher jitter peaks. This can be seen in graph 7.8. As Cubic does not react as fast as the other two to sudden large changes of bandwidth we get prolonged jitter periods. The reason for the big spike for the other two algorithms may be to the aggressive reduction of the bit rate. As seen the target bit rate goes to its minimum a lot faster than Cubic and this sudden large change requires the rate controller to adjust itself heavily. This may cause it to send frames a little bit later, causing the extra jitter peak. From a QoE perspective it is hard to see what is actually the better choice due to other variables such as target bit rate and round trip time. But we can conclude that when we have longer jitter periods the quality of the video is also worse. This indicate that it is okay to have some jitter peaks if it reduces the overall jitter periods.

8.2 Lund tests

In order to get some extra traffic in these tests we opened up a phone call via Skype to each other, used Spotify, ran two YouTube clips, a Netflix film and used Grooveshark at the same time. Even with this extra traffic every test except the alternating worked more or less perfectly. We had close to no packet loss and the video and audio was of very good quality. As the apartments are not to far from each other it was an expected result. We did however, also as expected, see an increase in jitter and round trip time compared to the local tests. This had however no significant impact on the QoE as they still were very small. What is interesting to note is that the random packet loss that occurred with the switch did no longer occur, indicating it is something with the switch. The alternating test showed quite bad quality when movement happened. To get some extra movement in the test we waved a hand in front of the camera as the other tests ran close to flawlessly. This caused a lot of video freezes, but no loss of audio. When looking at the statistics we could see that we got a lot of package losses but not very often in bundles above 10, seen in graph 7.16. As mentioned before we put a limit when the algorithms should actually adjust the target bit rate. In this case it may be so that we actually should remove or decrease this limit as we we probably want to adjust even when we have under 10 packet losses. With lower target bit rate it may have been so that the freezes disappeared as we would also send with a lower bit rate, making it more likely for the packets to not be lost. This is some fine tuning

that needs to be done and may be different depending if we are using the switch at AXIS or other network equipment. We could observe that Illinois performed better than Cubic and Cubic-Fit in the alternating test. This may be to the fact that Illinois starts at a lower target bit rate and as in this case the RTT is higher, more slowly reaches the maximum at the beginning. This can be seen in graph 7.15. Generally Cubic and Cubic-Fit work with a higher target bit rate as their implementation doubles the maximum compared to Illinois. When packet loss occurs above 10 they will almost never reduce the target bit rate under Illinois maximum, effectively running with max quality on the stream. With the removal of the packet loss limit both Cubic and Cubic-Fit will most likely use bit rates under Illinois maximum a lot more often, resulting in a better QoE.

In graph 7.19 we can clearly see how much impact motion or no motion has on the rate controller without adjusting the target bit rate. It becomes quite clear that the target bit rate is just a target and below that rate the rate controller has the freedom to do as it seems fit.

Interestingly all the algorithms pass the 1% requirements. Even if this was achieved the QoE was perceived as bad in the alternating test. A theory we have is that it might be I-frames that are lost instead of P-frames. As they are key frames and contain more information it will cause more lag, as explained in 2.1.3. This would explain why we got such heavy lag but so little packet loss. Looking at these tests it appears that Illinois is the better choice. We later did a test were we removed the PLL and we cold see that the QoE improved a lot. So much that all algorithm was hard do distinguish from each other and was of good quality. However this test was done without a second router, the second camera was directly plugged into the wall socket. The assumption we made in the local test chapter that it might be the switch's fault that we got seemingly random packet losses is thereby again strengthen.

Going back to the use of a delay based algorithm, we can see in 7.16 and 7.17 why this would not always work so well. We can see if we look at the round trip time that before movement even start we suddenly get a high round trip spike. This might be for example due to round trip noise. This would cause a delay based solution to react, but as seen in the packet loss graph we actually never get any packet loss at this point. Likewise we can see that we have a lot more packet loss than we have round trip time peaks, meaning a delay based might not react as often as we would like to.

8.3 USA tests

An interesting problem we had was that when we wanted to fetch a stream from here in Lund to USA, we got a GStreamer error concluding that we got data before we actually got a play request, causing no video and audio to actually be saved at all. When we instead delayed the camera by making it wait 2 seconds we instead encountered the problem a lot less. Decreasing this delay to 1 seconds caused the problem to occur more often. This may be due to a race condition where a play request is sent via TCP and some data via another TCP connection. As there is a large distance that must be traveled a packet may be delayed or lost on the way. This may cause the data to reach the camera in USA faster then the play request, especially if the play request is lost and need to be resent.

Another problem that we encountered when setting up a stream to or from USA was

that UDP traffic was blocked. This caused GStreamer to change to TCP transmission instead. Our algorithms currently only works on UDP based transmission and thus makes it impossible for us to evaluate how our algorithms work during a transatlantic call. We tried several things to work around the UDP block. These included things such as limiting which ports GStreamer was using and making sure these were not blocked by the router, making sure that the port forwarding was correct, giving the UDP more time before switching to TCP. None of this solved the issue. We encounter a similar problem already when we did the Lund tests. When we did our tests there we had an old router which blocked UDP traffic in one direction. Changing the router to a newer one solved this problem. So our final solution was to try to have the router in the USA changed. This was achieved and UDP was now forwarded correctly in both directions.

This leads us to conclude that a video conference application should implement a solution for both TCP and UDP. As some routers don't seam to want to let UDP traffic through but are okay with TCP, making an application without support of TCP may alienate a lot of users. This would also explains why Skype has an implementation that uses both protocols.

The low peak test was done when the internet traffic was at its lowest peak. This was done during 10-12 am in Sweden, or 04-06 am in Boston. During these test we got pretty much no packet loss at all as can be seen in 7.33, 7.34.

The algorithms ran nearly always at the maximum target bit rate in the alternating test 7.32. Due to more packet loss at the movement test the algorithm needed to adapt its bit rate more and we no longer could constantly run close to the max target bit rate. This can be seen in 7.31. As also seen, the bit rate was still quite high. The videos that was produced was of good quality and was perceived to have a good QoE. The movement videos did however have some lag in them due to still getting a few packet losses and doing large jumps in target bit rate. We could see that the RTT has increased since the test in Lund which was expected in 7.36 and 7.17. But as it still was very low it did not really affect the QoS or QoE at all.

When we ran the tests during peak hours, 8-11 pm in Boston, we immediately noticed in the videos that the algorithms had run with a lower target bit rate than in the low peak tests, as seen in 7.20, 7.21. We could see that this time around we got a lot more packet loss, seen in 7.23, 7.24. The total were still under or close to 1%, as seen, which is around the limit of video conferencing. But Cubic and Cubic-Fit kept its target bit rate to almost be at the lowest at all time. Illinois was also kept low and the fluctuation reappeared. As it continuously tried to poke, it would often get some packet loss, making it set the target bit rate low again. From a QoE perspective Illinois was perceived as worse than the others. As explained this fluctuation pattern is not good as it ruins the QoE, and even though Illinois generally had higher bit rate, this continuous change from lowest to a little higher target bit rate was a big distortion when watching the video.

The packet loss in these tests was 4-5 times higher than the test we ran at low traffic hours indicating that Netflix hours are a thing that must be taken into consideration when implementing a video conference application.

It is worth noting that the packet loss limit was removed for these tests. As the indication from the Lund test was that it was not needed, and may have even made the quality worse. In this test we can see that we do not get a lot of packet loss bursts that are larger than 10. In this case it might have been beneficial to have this limit, perhaps a little bit

reduced. This may have causes for example Illinois to not have the fluctuating pattern. On the other hand this might have simply caused more packet loss as the target bit rate would have been higher, and just as the Lund test caused a lot of freezes. Overall however, it appears that it is possible to have a video conference transatlantic with packet loss that is under 1%, with some small tweaks of some of the algorithms.

Looking at round trip time we could see that during all tests it almost always stayed below 150ms. Graphs 7.26, 7.27, 7.28 shows this clearly. Only a few times did we get a spike in round trip time, with the highest peak being just above 160ms. As expected, the round trip time was a lot higher compared to the Lund test, see graph 7.17. The conclusion we can draw from this is that in regards to round trip time a transatlantic call is very much possible. The requirement from mouth to ear was as mentioned before 150 ms. Since the RTT is below 150 ms and counts both the time between client and server and server to client, the actual network delay one way in our application is below 75 ms.

One very important point to discuss is synchronization. Even after an hour we still were in sync between audio and video. Even with higher round trip time and negligible more jitter than Lund, see 7.18, 7.29, 7.30, GStreamer were able to synchronize just fine, .

We could not do an real video conference as unfortunately our contact could not stay and help us do one. We did however have the USA camera filming a video clip of an interview. What we did instead was to use 3 streams. One stream recorded us in Sweden. Another was showing us what the camera in USA was filming via a web browser. Finally we had one stream that recorded what the USA camera was filming. One of the authors then pretended to be the interviewer and the other the one that was interviewed. That way we could sort of get an idea of the delay when we switched between who was talking as the view was switched completely, both for us and in the interview. The feed had to travel from the USA to us so we could change who was talking, and then back so the camera in USA could record what we was doing. We then stopped the streams at the same time, using this to be able to synchronize the videos recorded. This gave us an delay around 2 seconds in the switches. As it had to travel both direction, one direction would yield us around 1 second delay. If we then consider different delays in the camera we estimate that the real delay ranges from 200-400ms. This does not take in account the human reaction time for reacting to the switches which may further decrease this.

It is very important to note that this is not a very accurate measurement, but only an estimate. We would needed to have an actually person talking in the other end to properly measure the delay. It does however give us some idea of the delay. If the delay is closer to 200ms we would deem it to be acceptable. If closer or above 400 we would deem it not acceptable as it would detract heavily from the QoE.

An very interesting point from our video conference test is that Illinois had a very low packet loss rate. It could run with the highest target bit rate nearly for the whole duration of the call. This can be for a number of different reasons. One is very likely, and that is, as the previous test have shown, that during a video conference we have very low movement. This result in a very low bit rate, which may be fine to transfer transatlantic. But if that is the case, why would Cubic be so much worse and need to run with such low target bit rate, seen in 7.22? As shown in 7.25, only Cubic got consistently packet loss. One theory is that Illinois perhaps is not fair against Cubic. In previous video conference test we have used Cubic-Fit and Illinois and they have worked fine together. We actually never tested with Cubic as the articles stated that they are fair against other algorithms, such as TCP-RENO,

that only looks at packet loss. The Illinois article never explicitly states that is is fair against Cubic. In other words it might be so that Illinois hogs all the available bandwidth, causing heavy losses for Cubic. Looking at the target bit rate in graph 7.22 you can see that in the beginning Illinois does increase its target bit rate in an arc. At the same time Cubic decreases its target bit rate. When Illinois after a while quickly reaches the maximum target bit rate you can see that Cubic decreases its target bit rate a lot faster. Looking at this graph seems to point towards this being true, that Illinois hog all available bit rate. Another thing to consider is that we are actually sending two streams from the camera in the USA to the camera in Sweden. It could possible be that the camera in Sweden, running Cubic, need all the bit rate to be able to receive both streams, and thus could not actually send with a higher bit rate. Note that the web browser was displaying and requesting Motion JPEG. This streaming format requires a lot higher bit rate and peaks up to 30-40 Mbit has been noted, a lot higher than the H.264 stream.

8.4 Reducing the fluctuating pattern

One general problem was that when we have low bandwidth all algorithms still tries to probe the network to find the maximum bandwidth. Since we have low bandwidth we get a packet loss and instantaneously go down to a low target bit rate, this was especially clear with Illinois during the alternating test. This process is then repeated causing a fluctuating pattern which you can see in graphs 7.20, 7.21. The problem is that Illinois algorithm after a while of good conditions tries to go for the maximum allowed bit rate defined in the algorithm implementation. However the network might have seen good enough for the current bit rate of say 300 kbps but it will probe for a bit rate of 10000 kbps while the networks current capacity is actually 800 kbps. One way to counter this issue is to probe for a packet loss, then use this to determine a new maximum target bit rate that is set for a period and the algorithm is not allowed to exceed this rate. This will reduce the fluctuating pattern as we will never probe above where we got packet loss, making the overall stream more stable. If the maximum target bit rate is lower than the current bandwidth available it will most likely reduce the number of packet losses as we ensure we never over extends. This would for example probably help Illinois a lot over transatlantic calls. On the flip side, if we set such a limit on the maximum target bit rate for period and we suddenly get a lot of excess bandwidth, we will not be able to use this as the maximum will be below the now available bandwidth. One may try to combine the best of both, having the TCP congestion algorithms running with a maximum that another algorithm may be able to change as soon as it notice an increase or decrees in bandwidth. Due to time constraints and that we wanted to use as much as the bandwidth as possible this was never tried.

8.5 Performance

The cameras performance seems to not be a problem when we ran the tests. Even when the camera acts as an client and server at the same time the CPU usage is low. The only time it can be considered high is when we start a new stream, but even then the CPU available would be enough to start more streams. We could see that there was a slight difference

in running the server side and the client side application. As expected it was a little bit tougher to run the server side then running the client side. The difference however was not large, only a percent. We could also note that GStreamer background processes accounted for half or little more than half of the CPU usage when running either the server or client on one of the cameras. We first thought that the random packet loss came from an overload in the camera. But this seems very unlikely when we have looked at the usage of CPU. The problem is very likely in another area.

The memory grows during the execution at first but we could later see when running the one hour test that it stabilizes after a certain amount of time. We believe that it might be GStreamer allocating memory and not deallocating before it is necessary. This assumption was strengthen as we looked at the memory usage when we ran both the client and the server stream on the same camera. We could see that the memory usage was roughly the same and not the double as would be expected otherwise. As of now the camera should be able to handle several similar streams at the same time without any other performance enhancing optimizations being implemented.

8.6 Future work

There are many interesting things that can be made in the future. For starters our implementation only had peer to peer communication. In a video conference you may want to extend this to have more than just two peers. When having a conversation with multiple peers there is a need to implement a way to be able to show all peers at the same time and this is not currently supported in our application.

Currently there is no session initiation protocol, every stream setup is done directly in GStreamer. In the future you would like to have an easy way of setting up a conversation using a protocol like Session Initiation Protocol (SIP) to handle the communication between client and server.

What also could be done more thorough is the actual testing with regards to QoE. It would be nice to have more input than only the authors if the videos are okay or not. One may want to let users compare this solutions to others and see which one the users prefer. This brings us to an other important point:

At the moment we record videos to an SD card. It would be interesting to see if it instead was possible to let the camera display the person you are talking to live. At the moment you would need to record the video conference and look at it afterwards, which is not optimal. To be able to see the other partner live is an important feature in a video conference and thus is one of the most important future work to look at.

The algorithms themselves has a lot of different parameters that can be tuned. A fine tuning of these could be looked at. As an example the theta parameter in TCP-ILLINOIS, as have been already discussed, may be needed to be tuned depending of the network conditions.

As mentioned in the discussion it would be interesting to try to put a temporary limit on the maximum target bit rate to see if it would be more stable. This is something that may be of great value even if the throughput would be less as it may increase the QoE due to being more stable.

Currently the audio format G711 is used. OPUS is something that could be supported

on the camera and it should be better than G711. It lets you change the bit rate used on the audio at will, something G711 does not let you do. This would allow you to adapt the algorithms to also change the bit rate on the audio so we get better sound than G711 if we have high bit rate, or the opposite if necessary. At the moment we only change the video bit rate. We had some trouble with UDP to USA. Instead GStreamer switched to TCP. As our algorithms work by looking at packet loss, they would simply no longer do anything as we would never get any packet loss. It would be good if the algorithms, or newer and better ones, was implemented for TCP as well when UDP can not be used. The algorithms that we have used in this thesis is in theory used with TCP traffic, and an implementation of the algorithms where they work on both our UDP interpretation and the original TCP implementation should possible to do. The UDP variant could be run per default and if a router block UDP traffic the TCP variant could then be used instead.

A deeper investigation on where the random packet loss came from when running on the switch could also be made. The cameras clearly had the CPU and memory to handle the multiple streams. And since the algorithms used in this thesis had packet loss as a parameter to adjust it's rate, this caused a lot of disturbances in QoE for the user that should in practice be unnecessary.

8.7 Conclusion

It is clearly possible to set up a video conference using two AXIS network cameras, even though there are still a few things to do to get the QoE up on par to market standards. We could also see that algorithms that are based on both RTT and packet loss perform better than those only based on packet loss. The algorithm clearly was a better choice than no algorithm, if no algorithm was used the video feed only got through a few frames but with the algorithms the quality of the frames is reduced and a fluent stream can be watched. The previous master thesis had problems running two streams at the same time, however our implementation gives better results with two streams than on single streams indicating that our algorithms are fairer than the previous implementation. We also found out that an implementation that is supposed to support calls abroad must include an implementation of both TCP and UDP as a transport protocol due to routers blocking UDP traffic. Illinois tend to have a little bit better QoS than Cubic-Fit. But due to the fluctuating pattern, which is much greater at Illinois and Cubic-Fit being more stable, Cubic-Fit seams to be the better choice, at least to the USA.

Bibliography

[1] About Axis Communications AB. http://www.axis.com/corporate/index.htm, accessed 2014-09-04.

[2] Better Broadcasting with CBR. http://blog.twitch.tv/2013/02/better-broadcasting-with-cbr/, accessed 2014-12-01.

[3] Comparison of audio coding formats. http://en.wikipedia.org/wiki/Comparison_of_audio_coding_formats, accessed 2014-09-22.

[4] G.711. http://en.wikipedia.org/wiki/G.711, accessed 2014-09-22.

[5] Gstreamer application development manual (1.4.1). http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/manual.pdf.

[6] H.264/MPEG-4 AVC. http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC, accessed 2014-09-04.

[7] Opus codec. http://www.opus-codec.org/, accessed 2014-09-04.

[8] Performance Enhancements in the Next Generation TCP/IP Stack. http://technet.microsoft.com/en-au/library/bb878127.aspx, accessed 2014-09-04.

[9] Prominent features linux 3.2. http://kernelnewbies.org/Linux_3.2#head-1c3e71416a9fdc2f59c1c251a97963f165302b6e, accessed 2014-09-04.

[10] QoS Requirements of Video. http://www.ciscopress.com/articles/article.asp?p=357102&seqNum=2, accessed 2014-12-01.

[11] Real Time Streaming Protocol (RTSP). http://www.ietf.org/rfc/rfc2326.txt, accessed 2014-09-04.

[12] Real Time Streaming Protocoll. http://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol, accessed 2014-09-04.

[13] Real-time Transport Protocol. http://en.wikipedia.org/wiki/Real-time_Transport_Protocol, accessed 2014-09-04.

[14] Rtp: A transport protocol for real-time applications. https://www.ietf.org/rfc/rfc3550.txt, accessed 2014-09-04.

[15] Scalable Video Coding. http://en.wikipedia.org/wiki/Scalable_Video_Coding, accessed 2014-09-04.

[16] TCP congestion-avoidance algorithms. http://en.wikipedia.org/wiki/TCP_congestion-avoidance_algorithm, accessed 2014-09-04.

[17] Video compression frames. http://en.wikipedia.org/wiki/Video_compression_picture_types, accessed 2014-11-12.

[18] Clark D. Crowcroft J. Davie B. Deering S. Estrin D. Floyd S. Jacobson V. Minshall G. Partridge C. Peterson L. Ramakrishnan K. Shenker S. Wroclawski J. Braden, B. and L. Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet. 1998.

[19] Lawrence S Brakmo and Larry L Peterson. *IEEE Journal on selected areas in communications*.

[20] Julián Fernández-Navajas Luis Sequeira Luis Casadesus Carlos Fernández, Jose Saldana. Video Conferences through the Internet: How to Survive in a Hostile Environment. 2014.

[21] Cisco. Cisco vni service adoption forecast, 2013–2018.

[22] L. De Cicco and S. Mascolo. A mathematical model of the skype voip congestion control algorithm. *Automatic Control, IEEE Transactions on*, 55(3):790–795, March 2010.

[23] John William Evans and Clarence Filsfils. Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice. pages 8–13, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[24] John William Evans and Clarence Filsfils. Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice. pages 29–30, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[25] John William Evans and Clarence Filsfils. Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice. page 87, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[26] John William Evans and Clarence Filsfils. Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice. page 364, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[27] John William Evans and Clarence Filsfils. Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice. pages 22–24, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[28] Danny De Vleeschauwer Fernando Kuipers, Robert Kooij and Kjell Brunnström. Techniques for Measuring Quality of Experience. 2010.

[29] Behrouz A. Forouzan. Data Communications and Networking. pages [709–715], 1221 Avenue of the Americas, New York, NY, 10020, 2007. McGraw-Hill.

[30] Behrouz A. Forouzan. Data Communications and Networking. pages [715–735], 1221 Avenue of the Americas, New York, NY, 10020, 2007. McGraw-Hill.

[31] Behrouz A. Forouzan. Data Communications and Networking. pages [94,775, 913], 1221 Avenue of the Americas, New York, NY, 10020, 2007. McGraw-Hill.

[32] Davide Gerhard and Girardi Davide. Experimental analysis of the TCP Westwood+ and TCP CUBIC congestion control algorithms. 2012.

[33] Luigi A. Grieco and Saverio Mascolo. Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control. *SIGCOMM Comput. Commun. Rev.*, 34(2):25–38, April 2004.

[34] R. Frederick V. Jacobson H. Schulzrinne, S. Casner. Rtp: A transport protocol for real-time applications. 2003.

[35] Floyd S. Gurtov A. Henderson, T. and Y. Nishida. NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582. 2012.

[36] Wang Jingyuan, Wen Jiangtao, Han Yuxing, Zhang Jun, Li Chao, and Xiong Zhang. CUBIC-FIT: a high performance and TCP CUBIC friendly congestion control algorithm. *IEEE Communications Letters*, 17(8):1664 – 1667, 2013.

[37] Jingyuan, Wang and Jiangtao, Wen and Jun, Zhang and Yuxing, Han. TCP-FIT: An improved TCP congestion control algorithm and its performance. Tsinghua University, Department of Computer Science and Technology, Beijing, 100084, China, 2011.

[38] R. El Khoury, E. Altman, and R. El Azouzi. Analysis of scalable {TCP} congestion control algorithm. *Computer Communications*, 33, Supplement 1(0):S41 – S49, 2010. Special Issue: Heterogeneous Networks: Traffic Engineering and Performance Evaluation.

[39] King, R. and Baraniuk, R. and Riedi, R. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*.

[40] Kun Tan and Jingmin Song and Qian Zhang and Murari Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. Technical Report MSR-TR-2005-86, Microsoft Research, July 2005.

[41] Robert Kuschnig, Ingo Kofler, and Hermann Hellwagner. An Evaluation of TCP-based Rate-control Algorithms for Adaptive Internet Streaming of H.264/SVC. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, MMSys '10, pages 157–168, New York, NY, USA, 2010. ACM.

[42] Xu Lisong, K. Harfoush, and Rhee Injong. Binary increase congestion control (BIC) for fast long-distance networks. *IEEE INFOCOM 2004*, page 2514, 2004.

[43] Shao Liu, Tamer Başar, and R. Srikant. TCP-Illinois: A Loss and Delay-based Congestion Control Algorithm for High-speed Networks. In *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, valuetools '06, New York, NY, USA, 2006. ACM.

[44] Sally Lloyd. RFC 3649: HighSpeed TCP for Large Congestion Windows. 2003.

[45] Y. Cheng Google Inc. M. Mathis, N. Dukkipati. Proportional Rate Reduction for TCP, RFC 6937. 2013.

[46] Information Sciences Institute University of Southern California 4676 Admiralty Way arina del Rey California 90291. TRANSMISSION CONTROL PROTOCOL, DARPA INTERNET PROGRAM, PROTOCOL SPECIFICATION, RFC: 793. 1981.

[47] J. Postel. User Datagram Protocol, RFC 768. 1980.

[48] Xiaoyin Qi, Mianshu Chen, and Hexin Chen. A cavlc embedded method for audio-video synchronization coding based on h.264. In *Multimedia Technology (ICMT), 2011 International Conference on*, pages 16–19, July 2011.

[49] 3 M. Curado3 M. Yannuzzi1 E. Monteiro3 R. Serral-Graci'a1, E. Cerqueira2 and X. Masip-Bruin1. An overview of quality of experience measurement challenges for video applications in ip networks. 2010.

[50] Deborah Estrin Reza Rejaie, Mark Handley. Layered Quality Adaptation for Internet Video Streaming. 2000.

[51] Sandvine. Global Internet Phenomena Report. 2014. `https://www.sandvine.com/trends/global-internet-phenomena/`, accessed 2014-09-04.

[52] Ha Sangtae, Rhee Injong, and Xu Lisong. Cubic: a new tcp-friendly high-speed tcp variant. *Operating Systems Review*, 42(5):64 – 74, 2008.

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

[54] Szigeti, Tim and Hattingh, Christina. *End-to-End QoS Network Design: Quality of Service in LANs, WANs, and VPNs*. Cisco Press, 2004.

[55] Tibor Szkaliczki, Michael Eberhard, Hermann Hellwagner, and László Szobonya. Piece selection algorithms for layered video streaming in {P2P} networks . *Discrete Applied Mathematics*, 167(0):269 – 279, 2014.

[56] International Telecommunication Union. Recommendation ITU-T H.264, Advanced video coding for generic audiovisual services. page 3, 2013.

[57] International Telecommunication Union. Recommendation ITU-T H.264, Advanced video coding for generic audiovisual services. pages 451–452, 2013.

[58] International Telecommunication Union. Recommendation ITU-T H.264, Advanced video coding for generic audiovisual services. page 350, 2013.

[59] Wei, D.X. and Cheng Jin and Low, S.H. and Hegde, S. FAST TCP: Motivation, Architecture, Algorithms, Performance. *Networking, IEEE/ACM Transactions on*, 14(6):1246–1259, Dec 2006.

[60] Lai Yuan-Cheng and Yao Chang-Li. TCP congestion control algorithms and a performance comparison. *Proceedings Tenth International Conference on Computer Communications & Networks (Cat. No.01EX495)*, page 523, 2001.

Appendices

Appendix A

GStreamer launch examples

The following section shows and explains some examples of how we used GStreamer pipeline launcher command, `gst-launch`, to test experimental pipelines before we implemented and used them on the camera.

`gst-launch` combines GStreamer elements into a runnable pipeline by specifying the installed elements in the order that they are to be run and connecting them with an exclamation mark(!). This is a great feature if you want to see if certain elements works as intended without writing a full client and compile and run it on the camera. Here is a small example from the GStreamer reference manual

```
gst-launch filesrc location=hello.mp3 ! mad ! audioresample
! osssink
```

The following pipelines are more advanced and can be used on the video conference cameras directly to set up different streams. The first launch command sets up a video stream from the camera to a computer, decodes and shows the video stream live:

```
gst-launch-0.10 rtspsrc
location="rtsp://username:password@192.168.0.90/axis-media/media.amp?
resolution=1280x720" latency=0 ! rtph264depay ! ffdec_h264 !
queue ! ffmpegcolorspace ! xvimagesink
```

The next `gst-launch` pipeline sets up an audio stream between the microfon on one camera and the audio output on the other camera:

```
gst-launch-1.0 -v rtspsrc
location="rtsp://username:password@192.168.0.90/axis-media/media.amp?
video=0&audio=1&audiocodec=g711&audiosamplerate=8000&
audiobitrate=64000" latency=0 name=d ! rtppcmudepay !
mulawdec ! audioresample ! audioconvert ! audio/x-raw,
rate=32000, channels=1 ! alsasink
```

Finally we have a `gst-launch` pipeline that combines the two above and muxes both the audio and video stream into an file that is recorded on the SD card on the camera:

```
gst-launch-1.0 -v rtspsrc
location="rtsp://username:password@192.168.0.90/axis-media/media.amp?
resolution=1280x720&audio=1&audiocodec=g711&audiosamplerate=
8000&audiobitrate=64000" latency=0 name=d d.! rtppcmudepay !
mulawdec ! audioresample ! audioconvert ! queue ! matroskamux
name=mux ! filesink
location=/var/spool/storage/SD_DISK/cameratocameranewpipeline.mkv
d. ! rtph264depay ! h264parse ! mux.
```

Appendix B

Sample code

The following code is the same as the last `gst-launch` command in the previous chapter but written in C code instead. It works basically the same way, you first have to create all the elements, put them in a pipeline and finally linking them together.

```
GstElement *pipeline = gst_pipeline_new ("media-player");
 if(!pipeline){
    GST_ERROR("Failed to create pipeline");
}
GstElement *source, *audio_depay, *audio_dec, *audio_resample,
 *audio_converter,*audio_queue, *mux, *video_depay,
    *video_parse,
  *video_queue, *sink;
char full_url[1024];
char file_location[1024];
//Source element
source = gst_element_factory_make ("rtspsrc", "rtsp-source");
//audio elements
audio_depay = gst_element_factory_make ("rtppcmudepay",
   "mulawdepay");
audio_dec = gst_element_factory_make ("mulawdec",
   "mulaw-deccoder");
audio_resample = gst_element_factory_make ("audioresample",
   "audio-resample");
audio_converter = gst_element_factory_make ("audioconvert",
   "audio-convert");
audio_queue = gst_element_factory_make ("queue", "audio
   queue");
//video elements
video_depay = gst_element_factory_make ("rtph264depay",
   "h264-depay");
```

```
video_parse = gst_element_factory_make ("h264parse", "h264
   parser");
video_queue = gst_element_factory_make ("queue", "video
   queue");
//mux
mux = gst_element_factory_make ("matroskamux",
   "matroskamuxer");
//sink
sink = gst_element_factory_make ("filesink", "file output");

  //check if the elements could be created
if(!source){
   GST_ERROR ("source ");
}
if(!audio_depay){
   GST_ERROR ("audio_depay ");
}
if(!audio_dec){
   GST_ERROR ("audio_dec ");
}
if(!audio_resample){
  GST_ERROR ("audio_resample ");
}
if(!audio_converter){
  GST_ERROR ("audio_converter ");
}
if(!video_depay){
  GST_ERROR ("video_depay ");
}
if(!video_parse){
  GST_ERROR ("video_parse ");
}
if(!mux){
  GST_ERROR ("mux ");
}
if(!sink){
  GST_ERROR ("sink ");
}
if(!audio_queue){
  GST_ERROR ("audio_queue ");
}
if(!video_queue){
  GST_ERROR ("video_queue ");
}

// Setting rtspsrc properties
snprintf (full_url, sizeof full_url,
   "rtsp://%s:%s@%s/axis-media/media.amp?resolution=%dx%dy&audio=1"
"&audiocodec=g711&audiosamplerate=8000&audiobitrate=64000",
```

```
    name, pass, url ,width, height);
g_object_set (G_OBJECT (source), "location", full_url,
    "latency", latency, NULL);

  //Setting filesink properties
snprintf(file_location, sizeof file_location,
    "/var/spool/storage/SD_DISK/%s", file_loc);
g_object_set (G_OBJECT (sink),"location", file_location, NULL);

  //Add audio
gst_bin_add_many (GST_BIN(pipeline), source, audio_depay,
    audio_dec, audio_resample, audio_converter, audio_queue,
    mux, video_depay, video_parse, sink, NULL);

//Link audio elements
if(!gst_element_link_many (audio_depay, audio_dec,
    audio_resample, audio_converter, audio_queue, mux, NULL)){
  GST_ERROR("Failed to link audio elements: ");
  return NULL;
}
g_signal_connect (source, "pad-added", G_CALLBACK
    (source_new_pad), audio_depay);

  //Link video elements
if (!gst_element_link_many (video_depay, video_parse, NULL)) {
  GST_ERROR ("Failed to link video elements");
  return NULL;
}
GST_ERROR ("Link audio elements");
gst_element_link(video_parse, mux);
if(!gst_element_link_many (mux, sink, NULL)){
    GST_ERROR ("Failed to link mux elements: ");
    return NULL;
}
//connect elements that couldnt be linked at start of stream
g_signal_connect (source, "pad-added", G_CALLBACK
    (source_new_pad), video_depay);

GST_ERROR("Set state running");
gst_element_set_state(pipeline, GST_STATE_PLAYING);
```

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | 2015-03-25

EXAMENSARBETE Video Conferencing - Session and Transmission Control
STUDENTER Marcus Carlberg, Christoffer Stengren
HANDLEDARE Jimmy Rubin (Axis Communications AB)
EXAMINATOR Mathias Haage (LTH)

Videokonferens – Sessions- och transmissionskontroll

POPULÄRVETENSKAPLIG SAMMANFATTNING **Marcus Carlberg, Christoffer Stengren**

Videokonferenser blir allt vanligare i dagens samhälle. Vi kan se många applikationer så som Skype och Facetime på olika plattformar. Vi har implementerat videokonferensteknik på Axis Communications övervakningskameror och undersökt hur olika algoritmer som anpassar nätverkstrafik för att undvika stockning (eng:congestion), som till exempel TCP-Cubic, inverkar på kamerornas video- och ljudkvalité. Algoritmer har utsatts för ett antal tester för att utvärdera vilken som är bäst för videokonferens.

Introduktion
Tecken på en bra videokonferens är att kvalitén håller sig jämn och hög. För att uppnå detta måste så mycket av bandbredden som möjligt tas tillvara. Det är viktigt att paketförlusterna minimeras. Ett förlorat paket upplevs ofta som ett hack eller en total förlust av hela mediaströmmen av användaren. Hur uppstår då paketförluster? Det finns helt enkelt inte utrymme i nätverket för så många paket som man försöker skicka. När nätverket inte kan hantera mer så uppstår fördröjningar och paketförluster.

Stockningskontroll (eng: Congestion control)
Man måste anpassa mängden paket som skickas så att de får plats i nätverket. Algoritmer som gör detta utför stockningskontroll. I arbetet implementerades tre olika sorters stockningskontroll-algoritmer. Dessa är designade att anpassa överföringshastigheten av en ström av paket beroende på nätverkets trafiksituation. Dessa algoritmerna jämfördes i flera tester för att undersöka styrkor och svagheter.

Den första algoritmen (TCP-Cubic) baserar överföringshastighet på paketförluster. De andra två är så kallade hybrider, vilket innebär att de baserar överföringshastighet på två olika mått, i detta fall paketförluster och tiden det tar för ett paket att gå från klient till server och sedan tillbaka igen (eng: Round Trip Time, RTT).

Flera olika testscenarion sattes upp, bland annat lästes en Shakespeare-monolog upp. Det gjordes även tester över Atlanten till den amerikanska staden Boston för att se hur avstånd inverkar.

Slutsats
Att inte använda någon algoritm var inget alternativ. Då försvann i värsta fall all video och allt ljud då mer bandbredd än vad som fanns tillgängligt försökte användas. Det var också tydligt att hybridalgoritmer fungerar bättre. Hybriderna reagerar både snabbare vid sänkning av tillgänglig nätverksbandbredd och återgång till högre bandbredd. Vi kunde se att dessa algoritmer skulle gå bra att använda sig av i en videokonferensapplikation. Ett fenomen som inträffade, kallat fluktuering, var att i vissa fall ökade algoritmerna överföringshastigheten väldigt hastigt för att sedan minska direkt därpå. I dessa fall var det bättre att hålla en lägre men jämn kvalité hela tiden. Videokonferensapplikationen kunde leverera ljud och bild synkroniserat och fördröjningen var rimlig även till USA. Dock ökade paketförlusterna med 4-5 gånger under tidpunkter med hög trafik (kvällstid i USA, så kallade Netflix-timmar). Värt att notera är att paketförlusterna fortfarande är acceptable. Hade man även äldre routrar i nätverket så blockerades all UDP trafik vilket resulterar i att man måste implementera två lösningar, en för UDP och en för TCP för att kunna nå alla klienter.