Faculty of Science and Technology
Department of Computer Science

Reduce bandwidth in live-streaming by cooperative edge relay networks

Tor Seivaag Knudsen

INF-3981: Master thesis in Computer Science - June 2020

“Computers are like Old Testament gods; lots of rules and no mercy.”
–Joseph Campbell

“People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on each other, like a wall of mini stones.”
–Donald Knuth

Abstract

Live-streaming and particular live video-streaming are becoming increasingly popular as a multimedia-service [1]. More and more types of content and events are live-streamed. Live video-streaming has shown to be a useful tool in the case of pandemic outbursts such as Covid-19. As people should use social distancing [2, 3], live video-streaming can give people the ability to simulate conventional social contact.

Streaming in general, consumes much bandwidth. During the pandemic of Covid-19, several streaming providers had to reduce their streaming quality to meet the increased demand for bandwidth for their streaming content [4, 5].

This thesis present Cedge, a system that aims to reduce the massive amounts of bandwidth that live video-streaming use. Reducing bandwidth for live video-stream should also reduce the Internet Service Providers (ISPs) and content providers Total Cost of Ownership (TCO). Cedge aims to reduce the bandwidth by incorporating edge nodes in a Peer-to-Peer (P2P) overlay network, and organize client connections with different substructure architectures.

Our experimental implementation of Cedge shows that there are potential to reduce bandwidth with several order of magnitude for live video-streams.

Acknowledgements

I would like to give a huge thanks to my supervisor, Robert Pettersen, for all the time and hard work he has spent advising me. I would also thanks for all those educational and funny weekly meetings. Robert have given me countless good ideas, directions, and pointers in this thesis. I have been fortunate to get one of the best advisors, in my opinion, at the Institute of Informatics at University of Tromsø (UiT). It has been a great semester because of the support and encouragement that he has given me.

I would also like to thank Håvard D. Johansen, Dag Johansen, and the Institute of Informatics at University of Tromsø (UiT), for me and some of my fellow classmates to be able to join the Symposium of Operating System Principles 2019 conference in Toronto. The conference was very educational, and it was incredible to be able to join this conference.

I would also like to give my family a huge thanks for being supportive throughout these five years and earlier years.

Contents

List of Figures

List of Tables

List of Code Listings

List of Abbreviations

AAC	Advanced Audio Coding
AVI	Audio Video Interleave
CDN	Content Distribution Network
CPU	Central Processing Unit
DASH	Dynamic Adaptive Streaming over HTTP
DCT	Discrete Cosine Transform
FLV	Flash Video
FPS	Frames Per Second
HD	High Definition
HEVC	High Efficient Video Coding
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
ISP	Internet Service Provider
LAN	Local Area Network
LVS	Live Video-Stream

MKV	Matroska
MP4	MPEG-4 Part 14
OS	Operating System
P2P	Peer-to-Peer
QOS	Quality Of Service
RAM	Random Access Memory
RGB	Red, Green and Blue
RTMFP	Real-Time Messaging Flow Protocol
RTMP	Real-Time Messaging Protocol
RTMPS	Real-Time Messaging Protocol Secure
RTSP	Real Time Streaming Protocol
SBC	Single-Board Computer
SRT	Secure Reliable Transport
TC	Transmission Control
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TIL	Tromsø IL
TLS	Transport Layer Security
UDP	User Datagram Protocol
UiT	University of Tromsø
URI	Universal Resource Locator

VOD	Video on Demand
WAN	Wide Area Network

1 Introduction

Streaming is data that is sent over a network between different devices. Live-streaming is data that streamed in real-time. There are multiple forms of live-streaming, there is music, video, and other types of data that are live-streamed. In general, they are all live data-streams. The central part of this thesis will be about live video-streaming. More and more people use live-streaming in the form of live video-streaming [6]. Live-streaming is used in more types of content and events, for example, Twitter streamed more than 1300 live events through Q1 of 2018[7]. On Twitch, there is an average of 767 thousand concurrent channels that live-streamed in 2020, and an average of 1,92 million concurrent viewers in 2020[8].

Live video-stream creates a form of data that can be hard to put in a Content Distribution Network (CDN) since the data continuously updates in real-time. Transmitting live video-stream over the Wide Area Network (WAN) uses a large amount of data. Video-streaming, in general, is expected by Cisco to use 80% of all internet traffic by 2021 [9]. This could lead to many unnecessary data replications of the stream from the servers to the clients. This could also lead to pushing the hardware and optic fiber lines to the maximum. Which could increase the Total Cost of Ownership (TCO) to the Internet Service Providers (ISPs), and where the additional costs could propagate down to the clients.

The problem with the amount of data that is transmitted has been a significant problem for Mars/April of 2020 due to the Covid-19 virus outbreak.

Big platforms like Netflix and Youtube have tuned down the quality of their videos [4, 5]. They have reduced their quality because of the enormous load increase they have put on the WAN since large parts of Europe and America are under quarantine and social distancing.

1.1 Problem definition

The data in live-streaming is time-sensitive and should, in theory, be sent directly from the device that sends the original stream(source device) to the clients. However, the amount of data could be too much for a single device to handle. It would also waste many fiber line resources by sending the stream directly from the source device to all clients. Therefore, the live-stream could be sent through many types of servers such a CDN, before the stream reaches the clients. There will always be a balance between the latency and data utilization of the WAN.

However, there are untapped potential in the edge, close to the clients that could be utilized to reduce bandwidth usage. This leads to the following thesis statement:

A cooperative live-streaming application that incorporates edge devices in the Local Area Network (LAN) of clients, could reduce TCO of the streaming service provider, and reduce bandwidth usage on the Internet as a whole.

1.1.1 Use-cases

Live video-streaming are still very popular [1], and could be used in many different areas. One of the areas are Streamers which are people or groups of peoples that use a live streaming platform to distribute a stream. These Streamers could, for example, record themselves while playing games [10]. The Streamers could also commentate, discuss, or talking about a particular theme. Another area is live video-streaming of sports [6]. A match in a sport handles in real-time, and the live video-stream gives a very detailed view of the match in real-time. The third area for live video-streams is in large events. Since large events could also happen in real-time.

In general live video-stream seams to be used in specific areas that happen in real-time, such as events, sports, news, or entertainment. Live video-stream seams to be used in areas with capabilities to quickly change due to inputs, such as user inputs to a live discussion. Live video-stream does also have the potential to be used in video chats or online lectures [11].

1.2 Methodology

The final report of the task force on the core of computer science from ACM, presents a scientific framework to describe computer science [12]. The scientific framework presents three major paradigms of computer science. These paradigms are:

Theory

The first paradigm is theory and is rooted in mathematics. It further consist of four steps to ensure a coherent, valid theory. The steps are definition, theorem, proof, and the interpretation of the results.

Abstraction

The second paradigm is abstraction, which also could be referred to as modeling or experimentation. This paradigm is rooted in experimental science. This paradigm can also be further divided into four stages. Where stage one is form a hypothesis. Stage two is to construct a model and make a prediction. Stage three is to design an experiment and collect data from the experiment. Stage four is analyzing the results from the experiment

Design

The third paradigm is design, and it is rooted in engineering. This paradigm is further divided into four steps for constructing a system or a device that will solve a given problem. Step one is stating the requirements. Step two is stating the specifications for the requirements. Step three is to design and implement the system. Step four is to test the system.

The thesis is rooted in the abstraction and design paradigm as a thought of a problem formed into a hypothesis. From this hypothesis, a design for a system was created that contained some predictions. These predictions have potential limitations that can cause changes in the design of the system. Parts of the design was further implemented for experimentation of the system. Further requirements were needed to implement the system, and the architecture of the system was adjusted for the new specifications. A design for the experiments of the system was then created and executed. Finally, the collected data was analyzed and discussed.

1.3 Context

This thesis is in the context of the Corpore Sano Center [13] at University of Tromsø (UiT). The Corpore Sano center conducts research in multiple fields. The prominent fields are in computer science, sports, and medicine. The Corpore Sano Center has in-depth interactions between academics, end-users, and public stakeholders.

The Corpore Sano center develops fully deployed real-world systems and application prototypes. Some of the deployed projects are Bagadus [14] and Muithu [15]. Bagadus and Muithu are deployed at the soccer club Tromsø IL (TIL), which are one of our partners. Bagadus is an analytic application that uses sensors, annotation systems, and video processing on camera arrays. Muithu is a sports-analysis system that integrates real-time coach notations with related video sequences.

One of our key aspects in computer science is the work on security and privacy in systems. One of the work are secure abstraction with code capabilities [16], which embed executable code fragments in encrypted capabilities. Another work in security is Fireflies [17]. Fireflies is an overlay network protocol that organize members in a verifiable pseudo-random structure, to fight off intruders that try to take control of the overlaying network.

Another of our key aspects in computer science is the work on efficient collection and back-end storage systems. One of the works in this aspect of our group is Davvi [18]. Davvi is a video streaming system that provides search-based composition of video archive content. A client could request a keyword-based search to collect, index, and rank clips of a video, or videos into a single video, that contains relevant clips based on the keyword.

1.4 Outline

The rest of this thesis is organized as follows:

Chapter 2 specific information of different aspect with streaming and video in general that is relevant for the thesis, and the related work in this field.

Chapter 3 presents the design and potential architectures of Cedge. Discusses how the different architectures could affect Cedge.

Chapter 4 describes the implementation specific information about Cedge.

Chapter 5 presents the experimental setups of Cedge, and the results from the experiments are analyzed and discussed.

Chapter 6 concludes the design and result of this thesis, and outlines future work.

2 Technical background

The thesis stated that it used cooperative live-streaming to reduce the Total Cost of Ownership (TCO) and to reduce the bandwidth usage over the Wide Area Network (WAN). The central aspect of this thesis is the live video-streaming. This chapter contains valuable and technical related information that gives a more in-depth insight into the different aspects of this thesis. This chapter contains a description of a general design and architecture of a live video-streaming system. This chapter does also point to other systems that are related to Cedge.

2.1 Content distribution networks

A Content Distribution Network (CDN) is a server that helps distribute data that it receives from another server. The other server could be an "origin" server, a specialized server like storage servers, or another CDN. The CDN could for example be for caching data [19]. The CDN server is a geographical locale server that distributes data to clients that are geographical nearby. Someone could, to some degree, call a CDN an "ecological" server, since it "feeds" local clients with data.

The critical thing with CDNs is to reduce the amount of data that is sent between the origin server and the clients. This reduces the work for the origin server, the hardware costs, the latency for clients, and it could also increase the Quality Of Service (QOS).

Reducing the amount of data sent between the origin server and the clients could reduce the bandwidth usage on the WAN. The origin server could be far from the clients. The origin server could be in another country or another continent than the clients. If the origin server is far from many of its clients, then there is a possibility that much data is unnecessarily sent over long distances. This again could cause much unnecessary load of some parts of the WAN. By using CDNs, this could then remove the unnecessary data that is sent over WAN, and the network load of the WAN.

The CDN will handle the most amount of requests and data transferring between the clients and the origin server. The CDN tries to reduce the number of requests and data transferring between it and the origin server. This should reduce the network load on the WAN between the origin server and the CDNs, compared to the network load without CDNs. The CDN should take most of the requests and data transferring between it and the clients. This will not increase the network load between the CDN and the clients, compared to the network load without CDNs.

The CDNs handles more of the requests and data transferring of the origin server. This should then also reduce the work for the origin server. The origin server could then focus on pushing out new data to a "handful" of CDNs, instead of thousands or millions of users. The origin server could then also focus on other tasks like data aggregation.

The QOS could also increase by having an origin server and CDNs. If the origin server or a CDN for one area goes down, then the other CDNs could take over for the downed server. The CDNs might not be able to perform all requests if the origin server is down, but it should be able to do it if a CDN is down. As long if the other CDNs are not loaded to the maximum on the other clients. This will give the system/service a better uptime, thus a better QOS. Having more CDNs to take over if one of those goes down is not all the CDNs could do. If one of the CDNs gets a massive peak load, and cannot handle all the new sudden load. Then the other CDNs that are nearest could take some of those sudden load. The latency might be a bit worse when the other CDNs takes over some of the tasks, but its better than not be able to connect to the service.

The CDNs could also give a better latency for the clients. There could be less network routing when the CDN is geographically closer to the clients, which could lead to fewer network problems like packet loss. Since the CDNs takes over some of the requests and tasks to the origin server, that could mean the system is under lower stress. Lower stress could also mean fewer network problems like packet loss. Lower stress on the system could also mean that a CDN takes lower time to respond to a request. Lower latency for a system will also give a better QOS.

The CDN, as stated before, is a nearby server. This server could be from a small node to some nodes on a data center, or the CDN could be a large datacenter itself. This node or data center could have a variety of different tasks that they are specialized build to perform. Some applications that use a CDN could also have a variety of different nodes that it simultaneously uses together.

2.2 Video coding

This section of the thesis will only look at the video codecs and formats, and not other media codecs such as audio codecs. Video could be created by a camera or virtually with a computer. If it is not automatically compressed, then it is called raw video.

2.2.1 Raw video

Raw video is divided into frames. Each frame is then divided into pixels. Each pixel contains information about the light that the pixel should output. A pixel contains three bytes, one for each pure color. Other raw video formats could have different setups of the video structure, but this is a simplified version.

The pure colors in a pixel are green, blue, and red. Each byte contains a pure color value of 256 different color brightness. The first color brightness value is zero, and it makes that color transparent. The last color brightness value is 255, and that is the highest brightening of that color. If all colors have the lowest brightness value, which is zero, then the color is black. If all colors have the highest brightest value, then the color will be white.

If the color green had the brightness value of 255 and the other two colors had the brightness value of zero, then that pixel will only show a pure green light. If the green color only has a brightness value of 122, it would be a darker green light.

The color combining of pixels works to some degree, similar to mixing paint colors. If a bucket of red and green paint is mixed together, it will create a brown color paint. If the color brightness value of red and green is the same and above zero, and the blue value is zero, we will get a yellow color. This

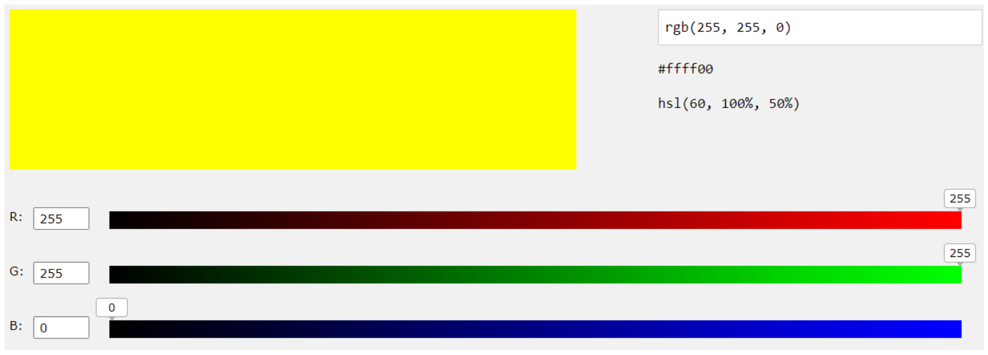

Figure 2.1: RGB Calculator

color mixing of pixels is displayed in Figure 2.1. The mixing of colors in paint and pixels gives the same effect, but different result.

The main problem with raw video is that it takes much storage to store a whole video. A single-pixel needs at least three bytes. Three bytes is not that much memory, but to put it in perspective, it could also store an integer that could be up to 16,5 million. Even if a pixel does not use that much memory, it will still use a considerable amount of storage. The central aspect of this is that we need to store many pixels in one frame, and a video needs many frames per second.

Storing a video which is in full High Definition (HD) needs 1080 pixels times 1920 pixels, and that is two million pixels for a single frame. The video could be running in 60 Frames Per Second (FPS), which means that the frame will be updated 60 times for each second of the video. That means one second of a full HD video of 60 FPS will use 124 million pixels. Those many pixels will use 378 million bytes, which is 378 megabytes per second. If the video lasts an hour, then that video will use 1,3 Terabytes of data.

Today someone could also get a video in 4K with 60 FPS, and it is in 3D. The 4K format uses 3840 x 2160 pixels. The 3D aspect means that it uses two video frames per frame. One second of a 4K 3D 60 FPS video use 3840 x 2160 x 2 x 60 = 995 328 000 pixels, which is three gigabytes. A whole hour of a 4K 3D 60 FPS video will use around 10,7 Terabytes.

To put this problem with the amount of data a raw video needs, into the perspective of sending it over the internet. If someone needs to send one second of the 4k 3D 60 FPS video over the internet, and both sides have 100 megabytes in upload and download speed, then it would take thirty seconds to send one second of that video. If a 4k 3D 60 FPS video was live-streamed over the internet, it could be very challenging to watch the live video-stream for the clients, if they could only watch one second of video every thirty seconds.

2.2.2 Video codec

A video codec is either a hardware or software tool that could compress or decompress digital video. There are two types of data compression. The first compression is lossless, and could also be called reversible compression. The second compression is a lossy compression, which could also be called irreversible compression. After data is compressed with one of the different types of compression, it would need to be decompressed before it could be used.

The lossless compression, compress the data so that all of the data could be recreated without any loss of data. While lossy compression, compress the data so that most of the data could be recreated. Not all of the data could be recreated by the lossy compression. The obvious drawback of using lossy compression is that it could not be restored to what it was, so there might be many areas where a lossy compression could not be used [20]. The drawback of using lossless compression is that it might not be as effective in reducing the total storage footprint as the lossy compression.

There are multiple different codecs that exist, and here are some of the codecs: H264 [21], High Efficient Video Coding (HEVC) [22], VP9 [23] and AV1 [24]. Each codec has different aspects, which gives them different areas where they should be used. The HEVC codec is the newer version of the H264 codec, and the AV1 is also the newer codec version of the VP9 codec. A potential drawback with HEVC codec, which is the new version, is that it is currently only for license use, which is explained in this license briefing [25]. The h264 codec does also have a license, but it seems to be cheaper and fewer royalty rates for small usage [26].

In general, these video compression algorithms split the video up in frames. These frames are then compressed with different compression algorithms. In general, there are three different types of compressed video frames called frame types. These frame types are usually called intra, predict, and bi-directional frames. For convince, they are referred to as I-frames, P-frames, and B-frames. Some codecs call them differently. For example, H264 referred to them as slices [27]. The difference between i-, p- and b-frames are displayed in Figure 2.2. The i-frame is a compressed frame that contains the whole frame. The i-frame is also the largest of the frame types in size. The i-frame is displayed in the low left corner of Figure 2.2. The i-frame is the first frame that is displayed in a video. The other frames will use it as a reference point, where they could update parts of the i-frame.

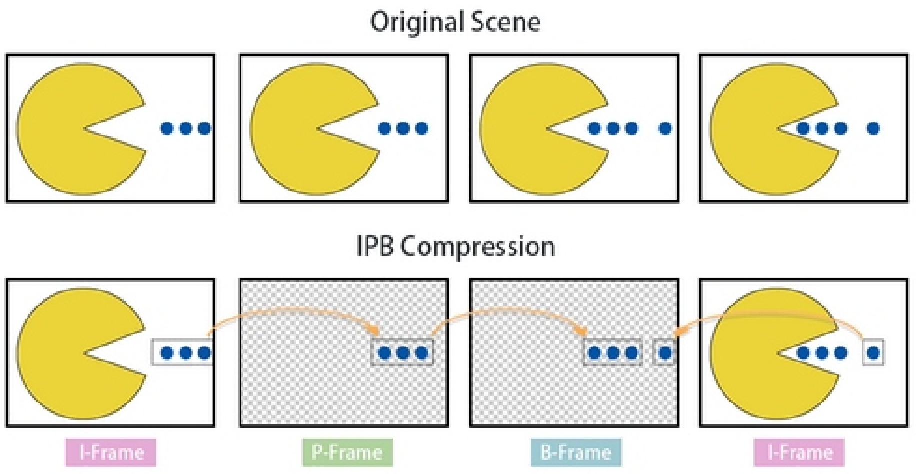

Figure 2.2: I-, P- and B-frames [28]

The p-frame is only a part or parts of the i-frame or the last frame. The p-frame might also contain a motion vector. The p-frame use the last frames to predict the next frame, and use the parts or vectors to update the old frame. This could be seen in Figure 2.2, where the dots that the player avatar eats are moved closer to the avatar. The three dots are cut from the i-frame and moved further to the center of the image. That means the p-frame could decompress the compressed i-frame and cut out those three dots. Then use the motion vector to move the dots closure to the center of the image.

The b-frame is very similar to the p-frame. The main difference is that the b-frame could both use parts of the last frame, and parts of the next frame that will be displayed after the current b-frame is displayed. The b-frame will combine the parts of the next and previous frames into a frame. This combining could be seen in Figure 2.2, where the b-frame first takes the dots that the player eats from the previous frame. Then take the dots from the next frame and combine it into the last i-frame.

The I-, P- and -B frames are divided further into smaller blocks, called macroblocks. These blocks will then take samplings from the surrounding pixels. The sampling will go through an algorithm to predict the image in that block. One of the algorithms could be Discrete Cosine Transform (DCT), which uses cosine functions to represent a macroblock [29].

The rest of this subsection will explore the h264 codec. The H264 codec works very similarly to the other codecs. The H264 codec will update the macroblocks instead of updating whole frames like i-frames or parts of the frames like p-

and b-frames. The H264 codec will also instead use SI-frames, SP frames, and multi-frame.

2.2.3 Video format

The video codec only compresses and decompresses the actual video. The video format is what the actual video itself is stored in. The video format could also be called a container. The reason why it could be called a container is that the format could also contain other multimedia such as sound or subtitles. There are many types of video formats such as: Matroska (MKV), Audio Video Interleave (AVI), Flash Video (FLV) and MPEG-4 Part 14 (MP4). A compressed video using the H264 codec can be stored in many different types of video formats.

2.3 Live video-streaming protocols

Live video-streaming protocols are protocols that could transport real-time video over the internet. There exist many different live video streaming protocols with different aspects, which should be used for different tasks. Some of the protocols are: Real-Time Messaging Protocol (RTMP), Secure Reliable Transport (SRT) and Real Time Streaming Protocol (RTSP). There is also some Hypertext Transfer Protocol (HTTP) based protocols that send chunks over HTTP. These HTTP based protocols are HTTP Live Streaming (HLS) and Dynamic Adaptive Streaming over HTTP (DASH).

These streaming protocols usually use two different underlying network transport layer to work. These underlying transport layers are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Each of these types has different aspects that are good or bad and could be used for different applications. Live video-streaming in the aspects of broadcasting a video from one to many clients should use the TCP transport layer. The reason to use the TCP transport layer is that even if a small part of the video frame is lost due to packet loss, then that packet could be resent. The stream could then look cleaner, and almost look like a video from Video on Demand (VOD). However, this does come at a price, the content providers that sent the stream might need to have a larger buffer, and the client device might also need to have a buffer of video frames. These buffers will then create latency, as the live video-stream first needs to fill up the buffers from the content provider and then in the client's local buffer before the client could then see the video.

When live video-streaming is introduced to video chat, then the whole video that is assured from the TCP transport layer, could become a drawback. The drawback is that the latency could be too much with the TCP transport layer. Then the UDP transport layer could be a much better alternative. The none assured packed loss in UDP, could assure whatever is left of the video after it is received, is then directly displayed for the clients. With UDP there should not be necessary to store some of the videos into buffers since they cannot resend the parts of the stream that was not received. However, the UDP transport layer might need a small buffer for video-audio sync. The main advantage with the UDP transport layer is that the latency could be very low while the apparent drawback of the UDP transport layer is that the video could be disruptive with a lousy internet connection.

There might be one main version of a live video-streaming protocol, and multiple different version of it. One of these live video-streaming protocols are RTMP. One of the version of the RTMP, is Real-Time Messaging Protocol Secure (RTMPS) which is RTMP with Transport Layer Security (TLS) encryption. Another version of RTMP is Real-Time Messaging Flow Protocol (RTMFP). RTMFP use UDP instead of TCP.

The live video-streaming protocols could contain more content than only the video. The live-streaming protocols should also be able to contain sound for the video, and some might be able to send multiple audio tracks. One audio track could be in English, and another audio track could be in Spanish, where a client could automatically switch between the languages. The live-video streaming protocols could also contain subtitles or multiple different subtitles.

2.4 Related work

Traditional live video-stream systems [30, 31] use different architectures, but have some similarities. The general architecture of a traditional system is visualized in Figure 2.3. In this figure, the source is pushing a Live Video-Stream (LVS) to the origin server. The source device could be a client that pushes the stream up to a streaming service like Twitch, YouTube, Instagram, or Facebook. The source device could also be a part of the internal system, such as a portable news camera that pushes the LVS up to the news site. There could also be numerous sources that each push a stream to the origin server. The origin server is the first server that the stream is pushed too. This origin server will push the stream to the CDNs, as shown in Figure 2.3. The origin server will also coordinate the CDNs. The origin server might also do some processing on the video of the stream before the origin server pushes the stream to the CDNs. The CDN is a server that could be geographically located near clients. The clients are shown in Figure 2.3 as red dots. These CDNs helps to distribute the stream from the origin server, and they also free up resources for the origin server.

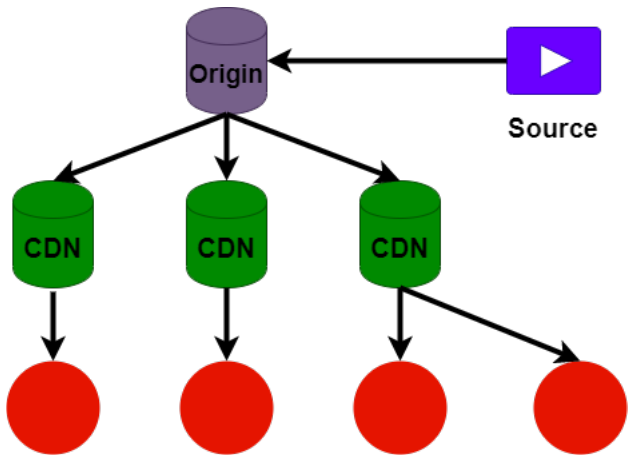

Figure 2.3: A traditional LVS architecture, the red dots are clients.

There has been much-related work that has already been done in the field of live video-stream. These systems usually have decentralized distribution architectures [32]. The systems usually use CDNs, relay-nodes or Peer-to-Peer (P2P) to distribute the live video-stream. Some systems might also use a combination of these distribution methods [33, 34]. Some live video-stream systems have a centralized organization architecture [31]. There could also be systems with decentralized organization architectures [30, 35]. Some systems have deployed edge nodes that are closer to the clients [36].

3 Design and architecture

The design and architecture of Cedge aim to improve existing live video-streaming systems. The architecture of Cedge will be modeled to combine two different approaches to distribution. The first approach is a cooperative network of relay-nodes. The relay-node is modeled to be an edge device inside the Local Area Network (LAN) of the client. The second approach is a dedicated server, such as an origin server placed on the Wide Area Network (WAN). These two approaches could then be combined into a hybrid distribution architecture.

3.1 Design of Cedge

The design of Cedge consist of four main components. These components are a *source device*, *origin server*, *orchestrator* and *relay-node*. The components can be seen in Figure 3.1. These components will be further explained in the next subsections.

3.1.1 Source device

A source device is a device that will feed the streaming service with a live video-stream. The rest of the components in Cedge will distribute this stream to all of the clients. The source device could be an internal part of the architecture

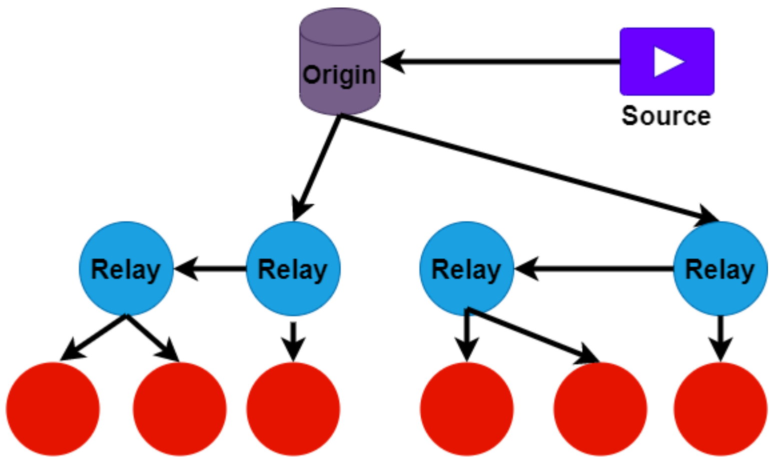

Figure 3.1: Overview of the Cedge LVS architecture. Relay-nodes streams here to multiple clients, the red dots are clients.

itself, where Cedge is in control of the device. For example, a news site with its own camera pushes a live video-stream onto the news site, where the camera is the source device. The source device could also be an external part of the architecture, meaning that the Cedge is not in control of the actual device. The external source device could be a client that pushes a live video-stream to the rest of the architecture of Cedge. There could also be multiple source devices that push numerous streams that will be merged into a single source for distribution in Cedge.

For simplicity, the source device will be referred to as a single device that pushes a stream to the architecture of Cedge.

3.1.2 Origin server

An origin server is located on the WAN and is the first receptor of the stream from the source device. The origin server will distribute the stream from the source device to the clients through the relay-nodes. The origin server must first know which relay-nodes should get the stream.

As the components of Cedge aims to improve existing live video-streaming services, it needs to distribute the stream through relay-nodes efficiently. Before the stream is pushed to the relay-nodes, the origin server must ask the orchestrator which relay-node it should push the stream towards.

There could be more than one origin server. These origin servers could work together in the same system or focus on a different part of the system. One origin server could, for example, be in the US and handle source devices and clients in the US, while another origin server could be in the EU and handle source devices and clients in the EU. For simplicity, the origin server will only be referred to as a single server, pushing the stream to the relay-nodes.

3.1.3 Orchestrator

The orchestrator could be a part of the origin server or be a separate device in the architecture. The orchestrator's primary mission is to efficiently distribute the stream to relay-nodes while not pushing the relay-nodes to their limits. The orchestrator will collect statistics from the relay-nodes, to determine which relay-node will be the first in a geographical area to receive the stream. The orchestrator will also be in charge of coordinating clients that leave and join.

3.1.4 Relay-node

A relay-node is an edge device that is located in the clients LANs. The relay-node will receive the stream from the origin server, or another relay-node, depending on how the orchestrator decides to distribute the stream. If the relay-node receives a stream from the origin server, it could transfer it to the client, or it could further push the stream to another relay-node. If the relay-node receives the stream from another relay-node, it will send that stream to the client, and it might push the stream further to another relay-node. When the stream goes through more than two relay-nodes, it will be referred to in this thesis as a relay-node chain. The relay-node could copy the stream and push it to more clients in the LAN without requesting more streams from the origin server. The relay-nodes will also provide the orchestrator with statistics that will help the orchestrator decide how to distribute the stream.

When the orchestrator decides which relay-node to push a stream too, it can only choose a relay-node that is located in the LAN of the requesting client. The origin server can not push data to a random relay-node since the relay-node is an edge device owned by the clients. The bandwidth the edge device use will be at a cost for the users of the LAN that the edge node is residing. It is the same principle if a relay-node push a stream to another relay-node, that relay-node which receives the stream, needs to have a client on their LAN.

There can be more than one relay-node in a single geographical area. One of the relay-nodes for an area could get the Live Video-Stream (LVS) from the origin server, and then it could push that stream to other relay-nodes in that area. This is to reduce the amount of work and bandwidth a single relay-node has to do. It is important to remember that the relay-node is an edge device that could have physical limits. The physical limits can be computational power, storage and memory, and bandwidth in both directions. These limits will be reported to the orchestrator to aid in deciding which relay-nodes to involve in the distribution of the LVS.

3.2 Architectures of Cedge

The first parts of a traditional live video-streaming system, described in Section 2.4, is similar to the architecture of Cedge. The source pushes the stream to the origin server as visualized in Figure 3.1. However, this is where the similarities stop. The origin server will push the stream to relay-nodes instead of Content Distribution Networks (CDNs). The relay-nodes will then push the stream to the clients. The relay-nodes can also push the data stream to numerous other relay-nodes, that will then serve the stream to the clients.

The architecture illustrated in Figure 3.1 is a layered architecture. Where the source device is in the first layer, the origin server and the orchestrator will be on the second layer. The relay-nodes will be in the third layer, and finally, the clients are in the fourth layer.

In the architecture of Cedge, the third layer itself can have numerous different substructures, which means that it will not affect the other layers, such as layer two and four. The different substructures in the third layer could affect the number of reduced data-links, latency, and the Quality Of Service (QOS) of the clients. There are some different substructures for the third layer of Cedge that will be described and discussed.

To discuss the various substructures for the third layer, we will use a real-world scenario to illustrate the uses better. The scenario we will use is a university class that generally use a big auditorium for lectures, that could fit 400 students, but is now unavailable. All 400 students need to use a digital platform to see the lecture live. It needs to be live so students can directly send in questions during the lecture and get an answer.

In this scenario, traditional streaming services will have a connection from the origin server to one client, referred to as one data-link. This scenario will also be a baseline for the resources needed for the origin server and the client. In this scenario, we will not use CDNs, but the result will be very similar if CDNs is used.

If the scenario had utilized CDNs, a data-link would instead be between the

CDN and the client instead of the origin server and the client, so the same amount of data-links would be used in both cases.

In our scenario, the live stream will use 2,5 megabit per second on average per stream. The 2,5 megabits are combined with the data that the Real-Time Messaging Protocol (RTMP) will use and the bitrate of the stream itself. 2,5 megabits multiplied by 400 students is one gigabit of upstream every second. The computer that pushes out the stream might not be able to push out one gigabit in upload speed. For our scenario, let say the computer has one gigabit of upload speed.

That means it could theoretically push out the stream for all 400 students, but it would most likely not handle all 400 students. If some frames have a huge frame change like an I-frame, that could cause the stream to peak at a higher upload speed. If the peak speed is higher than 2,5 megabit per second for each user, this will change the upload speed over one gigabit per second. Then that could lead to a network clog that would drop many frames for some or all users, and would also incur additional latency.

Another aspect of the problem of handling 400 streams with one computer is that it needs some computational power to handle all the streams. Then combine that with all the computational power, the encoding of the stream itself requires, which might be too much for a single computer.

3.2.1 Chain substructure

The first potential substructure of the third layer in Cedge is a relay-node chain. The relay-node chain works with an origin server pushing the stream to a relay-node, this relay-node will then further push the stream to a second relay-node and so on. Each relay-node in the chain can only push to one other relay-node, in addition to its clients. The relay-node chain will have a certain length of relay-nodes.

In our scenario, the professor's computer could act as a source device and maybe even the origin server with the orchestrator itself in one device. The orchestrator could then organize the relay-nodes that the students are connected to. In the scenario, some students could be studying together and be located on the same LAN. Here a single relay-node could serve all students on its LAN.

Now we could in theory, create an assumption of how much we could reduce the total upload speed from the above scenario. At first, we need to look at all of the students as a single LVS link. There are 400 students, and that means there are 400 links. Then we need to set the maximum relay-node chain length.

If we set the relay-node chain to a maximum of five relay-nodes, and each client is alone on their LAN. Then we could create an equation that gives the reduced bandwidth links in percentages:

$$1 - \frac{\left\lceil \frac{L}{C} \right\rceil}{L} \tag{3.1}$$

The "L" in Equation 3.1 stands for total LVS links. The "L" would in this example, be 400. The "C" in Equation 3.2 stands for the maximum relay-node chain. The "C" would in this example, be five. Notice the ceiling function around "L" divided on "C". This ceiling function is important to this equation as the links are a discrete number.

The equation will give a result of 0,8 with the example data of 400 students and a chain length of 5. This means 80% decrease in LVS links. The LVS links went from 400 and down to 80 links. The 80% decrease in LVS links will also mean an 80% decrease in the total average upload of the professor's computer. The professor's total upload speed would go down from one gigabit to 200 megabits.

If we take another example where some of the students are working together in a student group. Where the student group is connected to the same relay-node on the LAN. Also, the students are each watching the live video-stream of the lecture on their device. If there are eight student groups with ten students in each group, and the rest of the students are not in a group. Then we would need to update the equation too:

$$1 - \frac{\left\lceil \frac{L-GS+G}{C} \right\rceil}{L} \tag{3.2}$$

The "L" and "C" of Equation 3.2 is equal to the Equation 3.1. The "G" is the number of groups with more than one client that is connected to the same relay-node on the LAN. The "S" is the size of the groups. The LVS links to the students that are in a student group need to be added together and removed from the total LVS links. This would also mean that the number "G" of groups needs to be added to the total LVS links. So there would still be a single LVS link to each group after all group members are removed from the total LVS links. Then if we put in the numbers to Equation 3.2. Where "L" is equal to 400, "C" is equal to five, "G" is equal to eight, and "S" is equal to ten. The answer would be 0,836, which would mean that the decrease in LVS links are 83,5%. The total LVS upload speed from the professor's computer would go from one gigabit and down to 165 megabits.

The effectiveness of Equation 3.2 compered to Equation 3.1 is a reduction of stream links, but not a massive reduction. However, the reduction of stream links is not massive compared to the number of people that shared relay-nodes in Equation 3.2. It could look like an increase in the maximum relay-node chain is more effective in reducing the number of LVS links, compared to more devices that share a relay-node. However, both will reduce the total number of stream links. The Equation 3.1 and Equation 3.2 does not only work for this example but could also be used in general for Cedge.

3.2.2 Tree substructure

The second potential substructure of the third layer in the architecture of Cedge is a tree structure. This section will first look at a relay-node binary tree substructure. The relay-node binary tree substructure consists of one root relay-node. This root node will have up to two child nodes, where each child node could have up to two child nodes and so on. The root relay-node will get the stream from the origin server. Then the root relay-node will push that stream to its two child nodes. The child nodes of the root relay-node could then push the stream further to its children.

The relay-node binary tree substructure needs to have an absolute maximum height of child nodes. The height of a tree is the highest number of children's children. For example, if the tree only consists of the root node, it will be a height of zero. If the tree consists of the root node and two children, each has two children, then the height of this tree is two. This height is similar to the length of the relay-node chain. The maximum height of the tree is an essential feature because it could remove high amounts of latency for the clients. If the tree has a great height, then, the difference in latency between the root relay-node and the lowest leaf relay-node could be massive. That is also the reason for the importance that the relay-node tree must be able to self-balance. When the tree self-balance, it moves around the nodes to get the lowest height, and every node in the tree has both their children nodes. If the tree does not self-balance, then it could in the worst case, become very high in the height of one path of the tree and very short height in another part of the tree. This unbalance causes the clients in the high height path to get a higher latency than the clients in the short height path. The orchestrator will keep the tree in balance.

When there are more relay-nodes then what could fit in a self-balanced relay-node tree with a certain height, the orchestrator will initialize another tree for the rest of the clients. So if there are 70 clients and the maximum height of a tree is two, then there could only be seven nodes in a tree, therefor the orchestrator needs to initialize ten trees to put the clients in. A data-link in this relay-node tree substructure is from the origin server to the root relay-node of a tree, so in the case with 70 clients and the maximum height of two, there will be ten data-links.

To see how effective this relay-node binary tree substructure is, it would need to be compared to the relay-node chain substructure. Again we will assume there is only one client per relay-node. Then we will create an equation and compare it to Equation 3.1. This new equation for the relay-node binary tree structure will also use the example of a normal streaming service to see how many data-links it could remove. To simplify the equation, assume the orchestrator will be able to orchestrate all relay-nodes into a complete, balanced tree efficiently. If there are relay-nodes that could not complete a tree, they will fill a tree as much as possible. For this new equation, the maximum height will is set to five, which is the same length used in our example with Equation 3.1. Then find the total number "N" of relay-nodes a tree of height "H" could contain. This number "N" could be found with the same equation that output the maximum amount of nodes in a completely balanced binary tree. The maximum amount of nodes equation is:

$$N = 2^{(H-1)} + 1 \tag{3.3}$$

Then use the value of "N" in Equation 3.3 for the "C" variable in Equation 3.1. This new equation could then estimate the reduction of data-links with the relay-node binary tree substructure.

$$1 - \frac{\left\lceil \frac{L}{N} \right\rceil}{L} \tag{3.4}$$

The "L" in Equation 3.4 is the same as in Equation 3.1. The "L" is set to 400, "N" is calculated from Equation 3.3 with the height "H" of five. The result from Equation 3.4 is 98,25% reduction of data-links. The reduction of data-links will also mean a 98,25% reduction of upload speed for the origin server. The estimate of the upload speed from the origin server could now go from one gigabit down to 17,5 megabits of upload speed. The 17,5 megabits of upload speed are the theoretically lowest for Cedge with this substructure. In practice, the orchestrator and the system in general, would probably need an additional upload speed, to orchestrate the relay-nodes. The binary tree substructure could reduce the data-links with 18.25% more than the chain substructure in Equation 3.1.

To further reduce the amount of data-links, the height of the binary tree could be increased or increase the number of children in the tree substructure. M-ary trees are a tree structure that could have more than two children. The number of children each relay-node can have is decided on the number "M". A binary tree would, for instance have the number two as "M". M-ary trees do also use a root node for the first relay-node. M-ary trees also have a height number "H", which in this context will be used as a maximum height for the tree. It is also as important that the m-ary tree is self-balanced as the binary tree.

The number "N" from Equation 3.3 will be changed to incorporate m-ary trees. This new "N" equation will give the total amount of nodes in a perfect and balanced m-ary tree. The new equation for "N" is:

$$N = \frac{m^{(H+1)} - 1}{m - 1} \tag{3.5}$$

The orchestrator could then fill and balance all of the trees so that all of the trees could form multiple perfect m-ary trees. Then this new "N" in Equation 3.5 could be put into the Equation 3.4. If we set the number of children, "M", of Equation 3.5 to three, and the height "H" to five, the total data-links will be reduced by 99%. That means this specific m-ary tree could potentially reduce the data-links from 400 to only four data-links. This means that the origin server only pushes out ten megabits instead of one gigabit in upload speed. This example assumes that each client does not share relay-nodes.

This m-ary tree substructure for relay-nodes might be the most efficient way to distribute the stream to the clients when the efficiency is based on the number of data-links. An important aspect to consider before the m-ary tree substructure could be the most efficient substructure, is the average upload speed of the relay-nodes in the tree.

To find an estimate of the clients' average upload speed, then first look at the percentages of households in Norway that have broadband over fiber. The reason for looking at the percentages of fiber over broadband is that broadbands usually have the same download speed as the upload speed. However, there exists a hybrid fiber solution that has a higher download speed than the upload speed. In Norway, 54% of the total broadband subscribers have broadband over fiber [37]. One of the largest single broadband companies that deliver broadband to households in Norway is Telenor [38]. Telenor delivers four main different fiber subscriptions [39]. The slowest subscription Telenor delivers is 75 megabits in upload and download speed [39]. This could be used to create an estimate for the minimum average upload and download speed in Norway.

We will assume all clients have a minimum of 75 megabits of upload speed. In 75 megabits of upload speed, there could be roughly 30 full High Definition (HD) streams with 2,5 megabits of bitrate each. That will potentially mean that every relay-node with the m-ary tree substructure could push the stream to up-to 30 other relay-nodes. When each relay-node could push to 30 other relay-nodes, that means the "M" number of the Equation 3.5 will be 30. If the same height "H" of five is used in Equation 3.5, then the total amount of relay-nodes is an m-ary tree will be 837931. That means the origin server could have one data-link to the root node in the m-ary tree substructure, and the stream could then be potentially distributed to 837931 relay-nodes, which would mean that at least 837931 clients could watch the stream. However, this is only an estimate of the number of relay-nodes this m-ary tree could potentially distribute to. The orchestrator would probably use a lot of the upload speed to keep the m-ary tree balanced.

3.2.3 Hybrid substructure

The third potential substructure of the third layer in the architecture of Cedge is a hybrid version. This hybrid version could use both of the substructures that are described in the subsections above. It could also use or combine other types of substructures. The reason for having this hybrid substructure is that the relay-node chain and the relay-node tree substructure have different properties, that could be better for different scenarios of distribution of the stream.

The chain substructure is assumed to be a simpler substructure to handle, both for the orchestrator and for the relay-node itself. It might be easier for the orchestrator to follow the relay-nodes in a chain form since each relay-node only pushes to one other relay-node. If the third relay-node in a chain stops watching the stream or unexpectedly quits, the chain could easily be shortened by the second relay-node starts pushing the stream to the fourth relay-node. It might be better to have a chain substructure for the relay-node itself and the clients of that relay-node. The reason for that is the LAN that the relay-node is in could have a meager upload speed. If the relay-node uses most of the upload speed, it could affect other users of the LAN, giving them a bad experience.

As mentioned before in the subsections above, there could be multiple instances of the chain or tree substructures. These instances could also have different variables. The variables of the substructures are length, height, and number of children. The variables could also be changed for the instance of the substructures in different geographical areas but still be in one system. That means there could be a relay-node chain substructure in one geographical area with the length of ten nodes per chain. While in another geographical area, the relay-node chain substructure only have the length of three nodes per chain. The reason for different variables for different areas is that there could be, for example, some small communities with some clients on different LAN. If each community has a low upload speed and a low latency, then a longer chain substructure could be more efficient for that community. While in another community or area, there could be clients with a low upload speed but also higher latency between each other. That community might benefit more from chain substructures instances with a lower length.

The same with the tree substructures, wherein one geographically area a tree instance could have the height of six, and the number of children could also be six. In another area, the tree instance could only be a binary tree with a height of four. Both of the instances from these two areas could be in the same system and serve the same stream. The reason is also very similar to the different variables for chain substructures. One area could be a large city, with many clients closely connected. The connection speed could also have a very high upload speed and a low latency. Then an m-ary tree substructure could be used, with many children and colossal height. Where in another area that has fewer clients could have more tree instances with a lower height or the number of children.

Another aspect to consider when choosing a chain or tree substructure with different variables is how clients or user groups of the stream behaves. If the streaming services have statistics of its clients, it could organize clients from an area that often leaves early during the stream into a chain substructure. The reason to organize those clients into a chain substructure is that it might be easier to handle the rest of the clients when a client leaves the chain, while clients who have a great watch time could be organized into a tree since it might be harder to order and to balance the tree if a node leaves.

3.3 How Cedge orchestrate the clients into an architecture

The distribution of the stream with cooperative clients needs to be very efficient and well organized. As explained in the above section in this chapter, the component, which is called the orchestrator, will efficiently organize the clients. For more precision, the orchestrator does not need to organize the clients. However, it needs to organize the relay-nodes to the clients.

The orchestrator could keep a list of all clients that are connected to the streaming services at that moment in time. Which means the list will only keep clients that are currently watching the stream. Clients that quit the stream will be removed from the list. When a client wants to watch a stream, the client sends a request to the origin server or the orchestrator. That new client will be added to a list of clients. This list will have different information about the relay-node to the new client. The information of the relay-node could be about the broadband speeds to the WAN, the number of clients, the latency between the origin server and the relay-node, the location of the relay-node and hardware specs to the relay node. There could, of course be more information about the relay-node or the clients that are connected to that relay-node in the list.

The orchestrator could then use the list to create groups of relay-nodes and put them together into a substructure of the architecture in Cedge. To group the relay-nodes, the orchestrator would first need the location of the relay-nodes. The location of the relay-node could be found with IP-based geolocation. IP-based geolocation is mapping an IP address of a device to a geographic location. The location could be found with some different types of measurement techniques. For example, the measurement techniques could be the shortest ping to an IP address of a known landmark [40].

After the orchestrator has the information on where the relay-nodes are located, it could start to group the relay nodes together. For example, the orchestrator could take every relay-node that is located in a small city or community and create a group for these relay-nodes. Then the orchestrator needs to have the ping between the relay-nodes. The relay-nodes could then ping all of the other relay-nodes in that group and send them to the orchestrator. The orchestrator does now have useful information about the relay-nodes in this group. Then the orchestrator could start to create relay-node substructure instances, for example, some relay-node chains and some relay-node trees. The orchestrator could start with the relay-nodes with the shortest ping between the orchestrator and the relay-node itself. These relay-nodes could then be the head relay-node of the chains or the root relay-node of the trees.

3.3.1 Chain instances

When the head relay-node is established for the relay-node chain substructure instances, then the orchestrator will start to look for the other relay-node links in the chain. The orchestrator should add an additional relay-node link for each instance, and not fill a chain instance up before it starts with the next instance. This is to organize the chain instances more efficiently and prevent the chain instances from getting a very uneven latency. For example, if one chain is first filled to the maximum length and then start to fill the next chain, and so on. If the first chain instance gets all of the relay-nodes with fiber lines and is very close to each other, while the last created instance gets all of the relay-nodes with the slowest Internet connection, and the relay-node are widespread apart from each other. Then that could cause a massive difference in latency for the last relay-nodes in the chains.

The orchestrator might also create chains that have different lengths. To reduce the amount of time, it needs to organize the chain instances, or if a chain instance suddenly gets increased or reduced latency.

3.3.2 Tree instances

When the root relay-node is established for the relay-node chain substructure instances, before the orchestrator starts to fill the tree instances, it needs to have an additional step. That step is to check the bandwidth capabilities to the relay-nodes in that group. The bandwidth capabilities of the different relay-nodes could be used to create m-ary tree instances with different variables. For example, if there could be one m-ary tree instance with a maximum height of seven and the maximum number of children to five children per relay-node. Then the orchestrator could start to fill up the tree instances. It could be similar, as explained in the chain instance subsection. However, because of the different tree variables, there could be more focus on some particular tree instances.

3.4 Client join and exit in Cedge

The subsection of the hybrid substructure explains how the different architectures of Cedge could handle clients leaving the stream. A potential major problem with non-leaf/tail relay-node leaving is that there could be very hard to synchronize the stream after relay-node leaves accurately. For example, if the third relay-node link in a chain leaves during the stream, the second relay-node link will start to push the stream to the fourth relay-node link. However, that means the fourth relay-node link could miss out on many frames sent to the third link, and the third link does not send further down the chain. It also means the frame drop would propagate down the chain to the other relay-nodes.

The problem of synchronization and missing frames could be fixed with graceful leaving the stream. During a graceful leave, the relay-node could keep the connection open to the next relay-node link, while not sending it to its clients. The open connection could be kept until the stream is synchronized. The stream could be synchronized with the third relay-node link and the second relay-node link, both pushing the stream to the fourth relay-node link. The fourth relay-node link could then buffer some of the frames and maybe adjust the Frames Per Second (FPS) speed to create a smooth transition.

This synchronization problem is even worse for the tree instances. For example, there is an m-ary tree with ten children per node and a node in the second height of the tree that has ten children and leaves during the stream. Then the graceful leaving could potentially not be possible due to the number of children. The parent of the node that leaves could potentially need to push the stream out to its nine remaining children, and then start to push the stream to its ten grandchildren of its child that left. This is almost a doubling in upload speed for the parent and might be too much for the parent to handle. Which in turn, will most likely propagate to many frame drops for the nodes that were connected to that node, which left the stream.

If a new relay-node unexpectedly joins after the stream has started and the relay-nodes are in a group, and divided into instances of chain and tree substructures of the architecture in Cedge. The new relay-node could be added to a chain or tree instance if there is enough place in the chain or tree. If there is not enough place for the new relay-node, then a new instance could be created, and the new relay-node is then put into the new instance. The orchestrator might also see a potential to one of the instances or multiple instances into new instances that include the new relay-node.

3.5 Organize instances based on client information

The orchestrator could also collect and use information about the clients of the relay-nodes. The client information could be beneficial to organize the chain and tree instances efficiently. The reason for that is the orchestrator could better guess which client will leave early during the stream. As described in the subsection above, it is essential to have as little leaving as possible for both the chain and tree instances. There might be some clients that start many different streams and ends up leaving these streams shortly after they joined. The relay-node of these clients could then be put at the end of the chain instances or near the leaf relay-nodes in the tree instances. This to keep the potential frame drops from clients that leave to a minimal since then they will not affect as many other clients if they suddenly leave during the stream.

Another aspect that the orchestrator might need to keep track of is the number of clients connected to one relay-node. For example, if twenty clients that are connected to one relay-node, then this relay-node should maybe get a higher priority than the other relay-nodes with fewer clients. The reason for prioritizing is to create a better experience for as many clients as possible. The prioritization might then be used to put that relay-node closer to the origin server. So the relay-nodes with a higher prioritization could have lower latency, and maybe lower frames drop due to other relay-node leaving.

3.6 Discussion

The relay-node is an edge device, which is mentioned in the architecture of the Cedge section. However, what edge device should the relay-node be executed on. The edge device could be on the device that the clients use to watch the stream, but it should be on another device than on the client's device. The reason for that is the system in general, could potentially better handle clients that leave during the stream. Since when the client leaves or unexpectedly quits due to an empty battery on the client's device, then the relay-node on another device could still keep the rest of the chain or tree instance alive, while the device gracefully leave the stream. The edge device should be on a device that is always connected to the WAN. Some potential examples of what device could be an edge device is a smart TV, Chromecast, or Apple TV. One of the best devices for the edge device to execute on might be the home router. A potential additional feature arises when the edge device is located on the home router, that additional feature is lower latency for the clients and for the other relay-nodes that gets the stream from that relay-node. The reason for that is when the edge device is located on another device in the LAN, and then the home router will be the first device that the stream goes trough in the LAN. Then the edge device that is not on the home router needs to send the stream back and trough the home router again spread it to other relay-nodes. However, the latency inside the LAN should be very low, but when this repeats many times for each link in the chain instance or each child in the tree instance, then it could be noticeable.

4 Implementation

The thesis stated that a geographically cooperative streaming service could reduce hardware costs, and reduce bandwidth on the Wide Area Network (WAN). The first step to confirm the thesis statement is to test the central principles of the statement. The central principle of the architecture in Cedge is multicasting of the stream through relay-node chains.

The central principle of the architecture in Cedge is implemented in a simplified version. The next sections describe the implementation and deployment of the central principle in this thesis.

4.1 Relay-node

The relay-node is as described in Subsection 3.1.4 as an edge node. This edge device needs to be able to receive the stream from the origin server and to push the stream to clients. The edge node also needs to be able to push the stream to other edge nodes on other Local Area Network (LAN).

The edge node needs a server to receive and send the stream. The server should be light and efficient for the edge node. The server needs to be able to support live video-streams. The server would also need to support both the receiving the stream and sending the stream. The server would also need to be able to copy the stream and send it to multiple clients and other edge nodes.

The server that will execute on the edge nodes is called Nginx. The reason for not implementing a server from scratch is that it will be unnecessary time usage to create something that is already created. Nginx will be used to focus more time on testing the central principles of this thesis. Nginx is a highly used server for multiple different tasks [41]. However, Nginx does need a module plugin to be able to stream Real-Time Messaging Protocol (RTMP). This module plugin to Nginx is also a highly used RTMP streaming module [42].

The Nginx server with the module needs to be configured for it to work. There will be two different configurations of Nginx. However, they will be very similar, except for one line that can push the stream to another relay-node. This is the config for Nginx of the relay-node:

Code Listing 4.1: Nginx config

```
events {
    worker_connections  1024;
}
rtmp {
    server {
        listen 1935;
        chunk_size 4000;

        application dash {
            live on;

            dash on;
            dash_path /tmp/dash;
            dash_nested on;
            push rtmp://origin_or_relay-node_ip/dash;
        }
    }
}
http {
    server {
        listen 80;

        location /dash {
            root /tmp;
            add_header Cache-Control no-cache;
            add_header 'Access-Control-Allow-Origin' '*';
        }
    }
}
```

The configuration file is divided into three blocks, Events, rtmp, and http. The events block of the configuration file configures the global options that affect how Nginx handles connections [43]. The worker connections set the maximum number of simultaneous connections that can be opened. This number includes both clients and servers [43].

The RTMP block of the configuration file, configures the settings of handling all RTMP related connections for the Nginx server [44]. The server block inside the RTMP block defines a RTMP server instance. This instance will be the starting point for establishing RTMP connections. Once a connection is established, it will then be placed on another connection port. The server block also has an application block. The application block is a subsection of the server block, and there could be multiple applications within one server. The different applications could use different live video-stream format, such as HTTP Live Streaming (HLS) and Dynamic Adaptive Streaming over HTTP (DASH). The applications could also have different setups, such as live stream and live record the stream, pushing the stream to other servers, only allowing some to access certain content and executing other commands. The pushing of the stream to other servers is the feature that is the main aspect of this thesis, and it will be in every relay-node except leaf nodes or tail nodes.

The Hypertext Transfer Protocol (HTTP) block of the configuration file, configures how the application will handle HTTP and Hypertext Transfer Protocol Secure (HTTPS) connections [45]. The server block in the HTTP block works similarly to the server block in the RTMP block. The server block does also has a specified socket port for all HTTP connections. The location block is similar to the application block in the RTMP block. Both of these blocks could have multiple instances, and they are navigated to with a Universal Resource Locator (URI) or prefixes. They both divide the content served from the different protocols. The location block could also be nested [45].

4.2 Source device

The source device would need to be able to encode video and push the stream over RTMP to the origin server. Both the encoding and the stream transferring will be handled by a tool called FFmpeg [46]. FFmpeg is a highly used tool and has an extensive development with almost 100 thousand commits [47].

The content of the video itself should, in this thesis be static. The video could also be recorded or created before the video is streamed. The video should also be easily accessible for other people to recreate the implementation of this thesis and get very similar results.

The video in the live video-stream is pre-encoded in a format that supports live stream. The live stream format that the video is pre-encoded to is a Flash Video (FLV) container file format. The meaning of pre-encoding in this thesis is that the whole video is fully encoded before it is sent over the internet. If the video is not pre-encoded, it needs to be encoded while it is sent over the internet. Encoding while the stream is sent over the internet could take a considerable amount of data processing power. The amount of resources video encoding use depends intensely on the video itself. Some cameras do have in-camera encoding that could encode the video while it is being recorded. However, large manufactures such as Logitech have started to focus their attention on other aspects since now, nearly every computer offers dedicated video encoding chipset for efficient encoding [48].

The main reason for pre-encoding the already created video in this thesis is for creating a result that should be very similar if someone else tried to recreate the implementation of this thesis. This should give an excellent reproducing result. There are also other reasons for pre-encoding an already created video instead of streaming a video that is being recorded and encoded at the same time. One of those reasons is that the encoding and recording create extra latency for the stream. Another reason is that the type of content in the recording itself could create differences for the results, and would then be harder to reproduce the results. Another reason is that for full High Definition (HD), encoding could use a lot of processing power, which could then cause problems for older or slower computers. The problem for older and slower computers is that they might not be able to encode the stream in good enough quality in time.

There is two version of the video that will be used for evaluating Cedge. The first version is full HD (1920x1080 pixels) with 30 Frames Per Second (FPS), and the second version is 4k (3840x2160 pixels) with 60 FPS. The commands for encoding the video into a container file format that could be sent with RTMP is the first and second command in Code Listing 4.2. The two commands encode the two versions of the video as mentioned before the tool FFmpeg [46] is used for encoding both versions.

The "-i" flag in all of the commands is the video input, and following is the flag variable. The "-c:v" flag in the two encoding commands stands for encoding video into a format, and the format that the video encoded into is the flag variable, which is "libx264". "libx264" is a library for the h264 encoder. The "-preset" flag contains several different options for encoding speed to compression ration. The higher the compression speed is, the lower the quality of the video is. The lower the compression speed is, the higher the quality of the video is. The "placebo" flag variable to "-preset" is the lowest compression speed, which gives the highest quality [49].

The "-maxrate" flag is the limit of the maximum bitrate the stream could have. This includes both audio and video bitrate. The flag variable of the maxrate flag is 3000k, which means it has a maxrate of three megabits [50]. The "-bufsize" flag sets the size of the buffer to the stream. The flag variable of the "-bufsize" is set to 6000k, which is six megabits, which means that the buffer size is two seconds large. The buffer size is two seconds large because it is twice the size of the maximum bit rate per second. The "-pix_fmt" flag is the pixel format [51]. A pixel format is a color space that specifies how colors should be displayed. For example Red, Green and Blue (RGB) is a colour space. The flag variable of "-pix_fmt" is yuv420p.

The "-g" flag specifies the group of pictures together or "GOP". The "GOP" specifies the number of frames between i-frames [51]. The flag variable of "-g" is set to 120 frames in the 4k version, which means there should be one i-frame every two seconds. The flag variable of "-g" in the full HD version is set to 60 frames before each i-frame, which means there should be one i-frame every two seconds. The "-c:a" flag is similar to the "-c:a" flag, but it is for audio instead of video. The flag variable of "-c:a" is Advanced Audio Coding (AAC). The "-b:a" flag is a stream specifier for all audio streams. The stream specifier's flag variable is 128k, which stands for 128 kilobits in audio bit rate. The "-ar" flag sets the audio sampling frequency. The flag variable to "-ar" is 44100, which stands for 44100 hertz. In the first command, the "bbb_2160p_60fps_placebo.flv" is the output for the encoder, and the output is in a container file format called FLV.

The third command in Code Listing 4.2 will push the stream from the source device to the origin server. This command contains additional flags. The "-re" flag is a specialized input flag. The "-i" flag is very similar to the "-i", but it reads the input at a native frame rate. Which means that it could read one frame and then send it, and then read the next frame. While the "-i" flag will read all frames at once. The "-re" flag is a necessity for live-streaming a video. The "-c" flag contains the "-c:v" and "-c:a" flags, those flags are subsets of the "-c" flag. The flag variable of "-c" is "copy", which means that the "-c:v" and "-c:a" flags should not be encoded, but only copied from the input to the output. The "-f" flag forces the input or output formats. The argument flag is "flv", which means that it forces the output to be in FLV container file format. The "rtmp://relay-node-ip/dash" is the output of this command. The output specifies that it sent using RTMP protocol to a specified Internet Protocol (IP) address.

Code Listing 4.2: Commands

```
# 1. Command. Encodes a 4k video into a
# format that could be sent over RTMP.
ffmpeg -i bbb_sunflower_2160p_60fps_normal.mp4 \
-c:v libx264 -preset placebo -maxrate 3000k \
-bufsize 6000k -pix_fmt yuv420p -g 120 -c:a aac \
-b:a 160k -ac 2 -ar 44100 bbb_2160p_60fps_placebo.flv

# 2. Command. Encodes a full hd video into a
# format that could be sent over RTMP
ffmpeg -i bbb_sunflower_1080p_30fps_normal.mp4 \
-c:v libx264 -preset placebo -maxrate 3000k \
-bufsize 6000k -pix_fmt yuv420p -g 60 -c:a aac \
-b:a 160k -ac 2 -ar 44100 bbb_1080p_30fps_placebo.flv

# 3. Command, to start streaming the pre-encoded video
# to an origin server that supports RTMP.
# Stream command
ffmpeg -re -i bbb_1080p_30fps_placebo.flv -c copy \
-f flv rtmp://relay-node-ip/dash
```

4.3 Clients

The clients would need to be able to watch the stream that they receive from the relay-nodes. They would need something that could receive the stream, decode the video, and synchronize the audio to the video. A tool that comes with FFmpeg is called FFplay, and this tool could be used for receiving a stream, decoding the stream, and synchronize the audio to the video.

The command for watching a stream for the clients is displayed in Code Listing 4.3. This command uses the flag "-fflags" which specifies a flag variable for the ffplay tool. The flag variable is "nobuffer" which means that the stream will not use a buffer for the incoming stream. The "nobuffer" flag variable will reduce the latency that is created by the buffer. The "rtmp://relay-node_ip_addr/dash" is the input to ffplay. Which specifies a stream that use RTMP protocol from an IP address.

Code Listing 4.3: Commands for client

```
# 1. Command for start the stream on the client
ffplay -fflags nobuffer rtmp://relay-node_ip_addr/dash
```

4.4 Discussion

There is a lot of work that is all ready been done into the orchestrator, or something that works similar as the orchestrator [32, 52, 53, 54, 55, 56]. However, they are more focused on the Content Distribution Network (CDN) routing than relay-node routing. The origin server and the orchestrator will not be implemented since it is not needed for testing the central principle of this thesis. It would also be time-consuming and unnecessary to recreate components that already exist and are fully deployed into other systems.

5 Evaluation

The thesis stated that a cooperative close geographical streaming service could reduce Total Cost of Ownership (TCO) and reduce web traffic over the Wide Area Network (WAN). In this evaluation of Cedge, there are created experiments with different tests that will test the architecture of Cedge. The results from the measurements of the setups could prove that Cedge is a system worth exploring.

For all tests, there will be two different versions of the Live Video-Stream (LVS). Both versions are also pre-encoded on the source device. The versions are pre-encoded to h.264. The version is pre-encoded because the source device is not powerful enough to encode the video while it sends the stream. Even tho the video is both created and pre-encoded, it will still be sent over Real-Time Messaging Protocol (RTMP) as a normal LVS. This simplification is only to get a more precise test-result and to remove unnecessary variables from the experiments.

The first set of experiments will test the capabilities of an edge-node. It is essential to check whether a potential edge-node could work as a relay-node in the architecture of Cedge. The experiments will check the capabilities, and maybe the limits for this type of edge-node.

The second set of experiments will test the capabilities of Cedge. The architecture of Cedge is build up from several relay-nodes, and it is essential to see if they could work together to distribute the stream. It is also essential to check

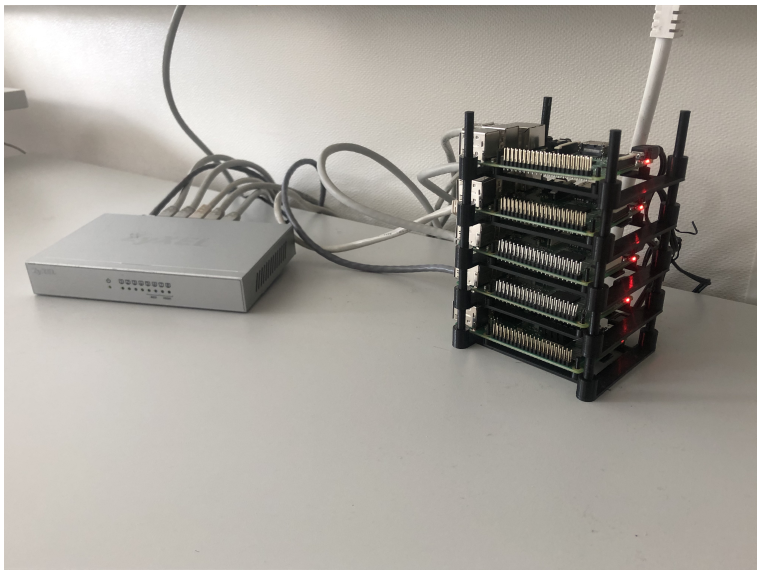

Figure 5.1: Relay-nodes

if there are other types of problems with distributing the stream with several relay-nodes.

5.1 Experimental Setup

The evaluation of Cedge was done with one source device, five Single-Board Computer (SBC); and 15 client devices. The source device has an 3.4 GHz quad core x86 i7-4770 Central Processing Unit (CPU). It has 16 GB of Random Access Memory (RAM) and 1000 Mbit Ethernet connection.

The five SBC are with two different types of SBC, three Raspberry Pi 3 model B, and two Raspberry Pi 3 model B+. The Raspberry Pi 3B+ have a 1.4 GHz quad core ARM CPU. They have 1 GB of RAM and 100 Mbit Ethernet connection. The Raspberry Pi 3 model B+ could have a maximum of 300 Mbit ethernet connection over USB2.0.

The Raspberry Pi 3 Model B has a 1.2 GHz quad-core ARM CPU. They have 1 GB RAM and a 100 Mbit Ethernet connection. The Raspberry Pi 3 model b and B+ are using a Raspbian Operating System (OS), that is based on the Debian OS. Each SBC will simulate a edge device where Cedge will execute on. All of the SBC are connected to one switch through an Ethernet cable, and the switch is connected to the Internet. The Raspberry Pi's and the switch could be seen at Figure 5.1.

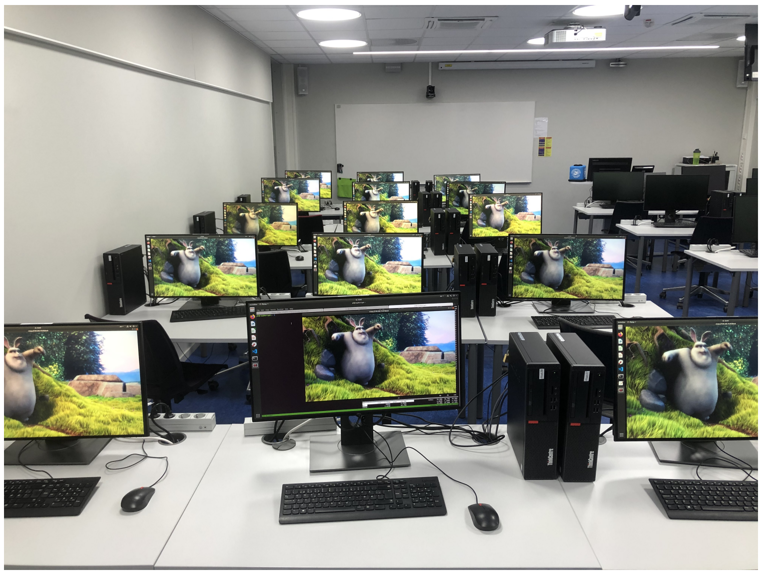

Figure 5.2: Illustration of the clients used in the experiment.

The clients has an 3.6 GHz quad core i7-7700 x86 CPU. It has 32 GB of RAM and 1000 Mbit Ethernet connection. The clients also have an 4k screen, which could be used for 4k content without the use of computer power to downgrade the LVS from 4k to 1080p or 1440p. The clients could be seen at Figure 5.2.

The video of the LVS is an animated cartoon called Big Buck Bunny [57]. This cartoon comes with many different scenes that have various aspects of the changes for each frame. The cartoon lasts for about ten minutes. There are two different versions of this cartoon for this test. Both versions show the same plot and have the same length. The only difference is the quality and size. There is one in 30 Frames Per Second (FPS) and full High Definition (HD)(1920x1080) screen resolution. The other LVS is in 60 FPS and 4k(3840x2160).

5.2 Capabilities of edge-nodes

The first experiment will both establish a baseline of how much resources a stream use, and how the latency will develop during a growing number of clients. This experiment will also show how a small device, such as a SBC, will be affected by a growing number of clients. This experiment will also establish the physical limitations of the number of clients that could be served by a single SBC. The SBC is an excellent choice for a small edge device since it has a low powered CPU and a small amount of RAM.

Table 5.1: Setups for capabilities of edge-nodes

SETUP	RELAYS	CLIENT/RELAY	VIDEO	TC
0	0	-	-	-
1	1	3	HD	-
2	1	6	HD	-
3	1	9	HD	-
4	1	12	HD	-
5	1	15	HD	-
7	1	15	4K	-
8	1	40	4K	-
6	1	15	HD	TC

The setup column of Table 5.1 is used to refer to the different setups of measurements in this experiment. The results for the different setups are displayed in Table 5.2. The first setup 0 in this experiment will find the "baseline" for hardware usage of a relay-node. This "baseline" will be further used and compared to the other measurements. Setup 1-5 of this experiment will measure the capabilities of a potential relay-node where the clients is a parameter that will increase for each measurement. Setup 6-7 of this experiment will try to determine the limits of the relay-node. Setup 8 of this experiment will measure capabilities in a simulated real-world for a potential relay-node.

The Table 5.1 shows different setups for the measurements in this experiment. The dash that is used in the rows of the table describes the not supported usage. The relay column describes the number of relay-nodes in the setup of the measurements. Table 5.1 only has two different setups of relay-nodes in this experiment. The relay row of the table with zero is a setup with no relay-node, but it has an edge device. The other rows with one relay-node do all have an edge device that is also a relay-node. The client/relay column describes the number of clients that are connected to a single relay-node. The video column describes the video quality that should be used for that particular setup. There are two different video qualities. The first one is in HD(1920x1080 pixels) quality with 30 FPS. The second video quality is in 4K(3840x2160 pixels) with 60 FPS. The last column named TC describes if that particular setup use Transmission Control (TC) to simulate a real WAN on the Local Area Network (LAN).

Baseline measurement

Setup 0 from Table 5.1 in this experiment will measure the relay-node without any load, or Nginx. The only load that is on the relay-node is a hardware measurement tool called HTOP. The relay-node will also be rebooted before the measurement of this setup.

The results of the measurement with setup 0 are shown in Table 5.2. The RAM utilization is 110 megabytes, and the CPU is 0,13% utilized. However, this test will not create a baseline for latency, average upload speed, average download speed, or peak upload speed.

Setup 1-7

The measurement of setup 1-5 in this experiment will use one relay-node, a source, and a great number of clients. The setups will look very similar to the Figure 5.3, where one relay-node casts the stream out to several clients. The source will send a LVS to the relay-node. The relay-node will push that stream to the clients. The clients are a parameter that will increase from three to 15, with the increment of three clients. This will show how the relay-node that is a SBC handles a growing number of clients. The CPU, RAM and network bandwidth will be measured on the SBC. The average latency will be measured on the clients. The variation of these measurements is also measured to see peaks that could be a potential problem. The start-up time will also be measured. Start-up time is from when a client sends a request to the relay-node, and the stream starts to play on the client's device.

In the measurement of the setups 1-5, the source machine, the relay-node, and the clients will be inside a LAN. All components of this measurement will also be connected with an Ethernet cable, which means there will be no connected devices with Wi-Fi. If the devices are connected with Wi-Fi, then there could be more latency or other problems. However, this will create an unrealistic low latency when every device is connected with Ethernet and is inside the same LAN. The other related network issues like packet loss, corruption, duplicates will also be unrealistically low.

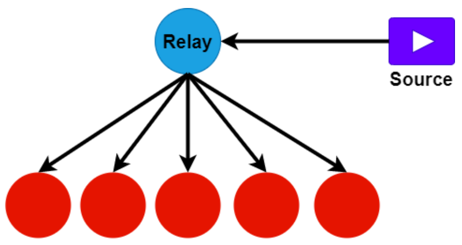

Figure 5.3: Architecture used to test the capabilities of the SBC. The red dots are the clients.

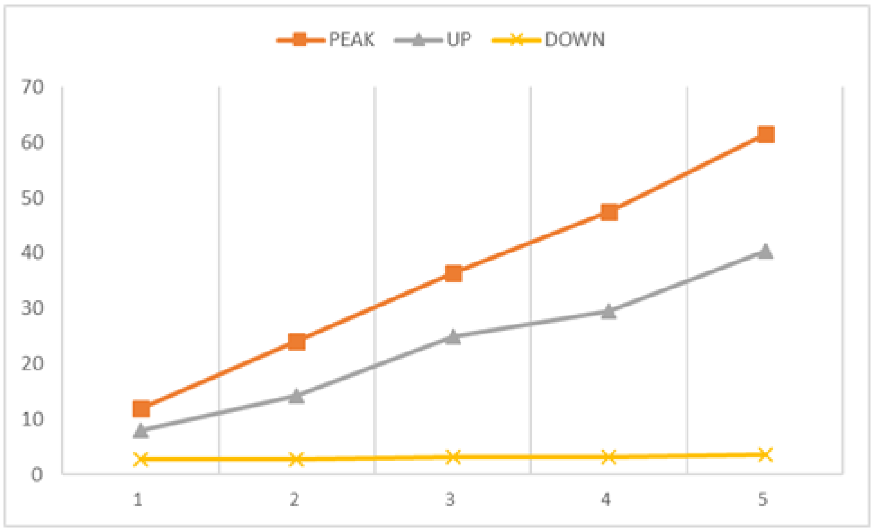

Figure 5.4: Bandwidth for setups 1-5. The Y-axis is in megabits

The results from setup 1-5 is shown in Table 5.2. These particular setups show a minimum latency of 462 milliseconds. The minimum latency is from setup 5. The highest latency is 1150 milliseconds, from setup 3. The average latency for all these setups is 722 milliseconds. The lowest utilization of RAM in these setups is 119 megabytes, and that is in setup 1. The highest utilization of RAM in these setups is 123 megabytes, and that is in setup 4. The average RAM utilization is 121 megabytes. The lowest utilization of CPU in these setups is 0,14%, and that is in setup 1. The highest utilization of CPU in these setups are in setup 4, with 0,45% utilization.

The bandwidth measurements of the setups are visualized in Figure 5.4. In Figure 5.4 The lowest peak upload speed for these setups 1-5 is 11,9 megabytes in the first setup. The highest peak upload speed is in setup 5 with 61,5 megabytes. The lowest average upload speed is also in setup 1 with 7,99 megabytes. The highest average upload speed is also in setup 5 with 40,4 megabytes. The lowest average download speed in these setups are in setup 2 with 2,72 megabytes. The highest average download speed of these setups are in setup 5 with 3,67 megabytes.

Setup 6 will see how the system performs under a heavier load. The normal LVS is a full HD(1920x1080 pixels) with 30FPS, and it will be switched to a 4k(3840x2160 pixels) with 60FPS LVS. This new stream should, in theory, push the system eight times more for each client than the former LVS. These measurements will be without any changes to the network card. Both of these measurements will be with only one relay-node.

The measurement of this setup will be without any changes to the network card of the relay-node. The relay-node will start to push a 4k LVS to 15 clients. The latency from the source device will be measured. The hardware components of the relay-node will also be measured to see how it handles under more load. The start-up time will also be measured.

The results of the measurement from setup 6 are in Table 5.2. The latency for this setup is 1382 milliseconds from source to client. The utilization of RAM is 125 megabytes. The CPU utilization is 0,32%. The upload peak speed is 56.9 Mbit/s. The average upload speed is 47.7 Mbit/s, and the average download speed is 4.22 Mbit/s.

Setup 7 will further push the limit of a relay-node. Setup 7 will be without changes to the TC on the network card. The meaning of this setup is to serve more clients than what is physically possible. This setup will only focus on the physical capacity of the Ethernet bandwidth of the relay-nodes. As explained in the experimental setup, the relay-nodes have only a 100 megabit ethernet connection.

Setup 7 will have 20 clients connected to one relay-node. Each client has two instances of the stream. That means each client will get twice as much data from the relay-node. The LVS will be the 4k with 60 FPS, and this stream has a bitrate of around 3,1 megabits per second. The total amount of data the relay-node need to upload to the clients is around 124 megabits per second. Both versions of the relay-node have a 100 megabits Ethernet connection. So the relay-node will have a problem to distribute the stream to all of the clients.

It will be interesting to see if the relay-node will start to use a lot of a particular hardware component. Will it, for example, try to cache more of the stream in the RAM. How much data will it be able to push out to the clients, is it 100 megabits or could it only be 85 megabits. It will also be interesting to see how the relay-node will handle the clients. Will it try to only send a full stream to some of the clients. Perhaps instead, it tries to send the stream to all clients. The first data packet is transmitted to some clients, and the next data packet is transmitted to the other clients. This setup might cause numerous frame drops.

The results of the measurement from setup 7 are in Table 5.2.

Setup 8

Setup 8 will be similar to the last setups. This setup also has one source, a relay-node, and a significant number of clients. This setup will also be inside the LAN, but it will use a tool to simulate a WAN. The settings for the tool TC in the Linux kernel will be changed to simulate a WAN in the LAN. The settings will only be changed for the relay-node. The change will give extra latency for all incoming and outgoing data transmission. The changes will only affect the network between the source device and the relay-node. The reason why the clients should not directly be affected by the network changes is that they are on the same LAN as the relay-node. The latency between the clients and the relay-node with Ethernet is then presumed to be more realistic with the network changes.

The relay-node will also be given a maximum network bandwidth to simulate the restrictions from a Internet Service Provider (ISP). The maximum bandwidth will also be between the network of the source device and the relay-node. There will be no maximum bandwidth between the relay-node and the clients. There is expected that there are no virtual or physical bandwidth limitations on the LAN, unlike the WAN, where the network has virtual bandwidth limitations from the ISP.

Even thou the estimate from Chapter 3 shows that the network bandwidth could be 75 megabits, there are still many users that use dsl [37], and they might have lower speeds. So instead, use an older estimate from 2017 where the bandwidth is 23.5 megabit [58].

The first aspect of simulating a WAN on the LAN is to find the average latency of the geographically nearby servers. The geographically nearby servers must also not be in the LAN that the test is performed. The Linux tool Ping is used to perform the latency tests for nearby servers. Ping is a tool that gives a round trip time of how long it takes to get a response from a server. Ping gives a time around 27 milliseconds for geographically nearby servers. The Ping time was also down to 22 milliseconds for some servers. To give a better aspect of the first test's problems, the Ping time between the relay-nodes was around 0,4 milliseconds. To make this more realistic, the added latency for the test is 27 milliseconds and with an additional random normal-distributed latency of +- 5 milliseconds.

The results of the measurement from setup eight are displayed in Table 5.2. The latency was 440 milliseconds. The RAM is 116 megabytes utilized. The CPU is 0,4% utilized. The upload peak speed is 54,8 megabits. The average upload speed is 39 megabytes, and the average download speed is 3,35 megabits.

Table 5.2: Latency and hardware results of edge-node

SETUP	TIME	LATENCY	RAM	CPU	PEAK	UP	DOWN
0	-	-	110MB	0,13%	-	-	-
1	131	0,563	119MB	0,14%	11,9	7,99	2,82
2	123	0,82	121MB	0,26%	24,2	14,3	2,72
3	150	1,15	122MB	0,38%	36.5	24,9	3,31
4	122	0,615	123MB	0,45%	47,6	29,6	3,11
5	172	0,462	122MB	0,4%	61,5	40,4	3,67
7	217	1,382	125MB	0,32%	56,9	47,4	4,22
8	114	3,525	144MB	0,27%	95,5	92,8	4,55
6	168	0,440	116MB	0,4%	54,8	39	3,35

The Table 5.2 shows the results of the first experiment. The first column regarding the setup that is further explained in Table 5.1. The second column of Table 5.2 is named time, shows the timestamp in the stream when the measurements were taken. The time column is in seconds. The third column is named latency, and that column shows the latency in seconds. The latency is between the source device and a client of the relay-node.

The fourth column is named RAM, and that column describes the RAM utilization of the relay-node. The RAM column is in megabytes. Furthermore, it shows the RAM utilization of the current time. The maximum amount of RAM a relay-node can have is one gigabyte. The fifth column is named CPU, and that column describes the CPU utilization of the relay-node. The total maximum CPU utilization will be displayed in this column as 4.00 %. 1.00 % is the full utilization of one core, and the relay-node has four cores. The CPU utilization percentages for this column is the average CPU utilization in a time-frame. The time-frame or CPU utilization is one minute.

The sixth column is named "peak". This column describes the highest upload speed that is recorded in the last 40 seconds. The peak is measured in megabits. The seventh column is named up, and it describes the average upload speed of the last 40 seconds. The up columns are measured in megabits. The last column is named down. This column describes the average download speed of the last 40 seconds. This column is also measured in megabits.

5.2.1 Discussion

In these setups, it is expected that some of the different measurements will in theory, create a linear graph growth. A twice as big increase of clients will cause a twice as big increase in utilization, and this linear graph curve is apparent in peak upload speed. This linear graph could be seen in Figure 5.4. The peak upload speed starts with almost twelve megabits with three clients and grows to around 24 megabits with six clients. It looks to be that the peak upload speed grows with twelve megabits for each time three clients that have been added. If the upload peak speed is divided among the number of clients, it gives a very similar peak per client for all setups. The peak per client varies from 3,9 megabits to 4,1 megabits. The setups 1-2 have both a peak per client of 3,9667 megabits. Setup 3 and 4 has a peak per client slightly over four megabits. Setup 5 has the highest peak per client of 4,1 megabits. The upload peak speed is dependent on the number of clients.

The average upload speed does almost look to also have a linear growth in Figure 5.4, but when the average upload speed is divided among the number of clients. Then it shows the average upload speed does have a noticeable variation but does still have a very linear growth. The smallest upload per client would be in setup 2 with 2,23 megabits, and the highest upload per client would be in setup 3 with 2,76 megabits.

The average download speed almost shows an increase from top to bottom. This increase is hard to see in Figure 5.4. So instead, look at the result for the setups in Table 5.2. However, the increase does drop down every other setup. For instance, the second and the fourth setup have a lower download speed than the setup before those particular setups. The increases look to be caused by more clients and streams with more data. However, this might occur because of how Transmission Control Protocol (TCP) works. The increased download speed might be the result of more clients that send acknowledgments to the relay-node. The increase could also be the result of more clients that might need to get a packet re-transmitted.

The other measurements of these setups do not seem to have a linear graph growth. The setup 1-3 looks to have a linear increase in latency and utilization in RAM, and CPU. However, setup 3-5 looks to have a decrease in latency and CPU utilization.

Table 5.3: Start-up time results of edge-nodes

SETUP	START-TIME
1	3,82
2	3,966
3	4,373
4	4,146
5	3,804

An interesting aspect for all setups in 1-5 is the upload peak speed compared to the average upload speed. The results from Table 5.2 shows that the upload peak speed seems to be around one and a half time bigger than the average upload speed. The upload peak speed is divided on the average upload speed to find an actual estimate. The result gives an estimate for the upload peak speed to be 1,55 times bigger than the average upload speed. The results do variate from 1,46 to 1,69 in the setups. The result is also only for the full HD stream with 30 FPS.

This theoretical estimate gives a small equation that could be used to find the upload peak speed.

$$P = 1,55A \tag{5.1}$$

Where "P" is the upload peak speed and the "A" is the average upload speed. This small Equation 5.1 could also be further expanded to find the theoretically maximum of clients that could be connected to a relay-node with a full HD stream:

$$C = \frac{E}{1,55A} \tag{5.2}$$

Where the "C" the number of clients that could be connected to one relay-node. "E" is the maximum upload speed of a relay-node, and "A" is now the average upload speed from one client. If this expanded equation is used for the relay-nodes in this experiment, the relay-nodes has an Ethernet maximum upload speed of 100 megabits. Then find the "A," which in this expanded equation is the average upload speed from one client. The results from setup 5 are then used to calculate "A". 40,5 megabits divided among 15 clients give the result of 2,69 megabits for each client's average upload speed. The result for Equation 5.2 is 23,98 clients. That means the theoretically maximum clients a relay-node from this experiment could have is around 23-24 clients with full HD stream.

In setup 5, the bandwidth increased up to a peak upload of 61,5 megabits per second. Almost two thirds (2/3) of the whole Ethernet bandwidth is used during the peak. If the results from Equation 5.2 is compared to setup 5, then setup 5 is only around eight to nine client away from the maximum limit of clients. The upload peak speed is an important aspect to keep track of in a live video-streaming service. If the peak goes above the total bandwidth, it could cause many frames to drop or terminate the connection to clients. This could be very frustrating in live-streamed sports, where the peak happens during critical moments of the game.

The 4k stream in setup 6 should in theory, use eight times more data than the full HD stream from setup 1-5. Setup 6 will be compared to setup 5, since setup 5 has the same number of clients and used the full HD stream. The average upload speed is divided on the number of clients in both setups. The result should give an estimate of the size difference of the stream. The result for setup 5 when the average upload speed is divided among the number of clients is 2,69 megabits. The result for setup 6 is 3,16 megabits. The data stream for setup 6 is at least more significant than in setup 5. However, it is not even twice as big in upload speed. The reason for such a low increase in upload speed could be the encoder. The encoder could potentially use fewer more extensive frame updates, which could reduce the upload speed considerably.

An exciting aspect from setup 6 is that the stream has a lower peak upload speed, than the other full HD stream with 15 clients from setup 5. The upload peak speed is 4,6 megabits lower than the full HD stream. However, the average upload speed for this measurement is 46,5 megabits, which is almost 1,5 times higher than the full HD stream's average upload speed. An aspect of the lower peak might also be the result of how the encoder encoded the stream. The encoder could also give a different time for the upload peak speed for the two different stream qualities.

The download speed on the relay-node is also one megabit higher than the full HD stream. The latency also increased considerably amount up to 1382 milliseconds in the measurement of this setup. The latency is 920 milliseconds higher than with the full HD stream. The CPU of the relay-node is down from 0,4% of use in the full HD stream to 0,32% in the 4k stream. The RAM of the relay-node is up with three megabytes to 125 megabytes in total.

Setup 7 is pushing the limits of a relay-node. This setup will be a good benchmark for the limitation of an edge device that could become a potential relay-node. Serving 40 clients give the biggest latency so far, with 3525 milliseconds. The upload peak is also the biggest so far, and it is up to 95,5 megabits. I would assume that the 95,5 megabits are the physically maximum limit for the relay-node. As described, the bitrate of the stream is 3,1 megabits. Take the bitrate and multiply it with 40 clients, and it should give an actual upload speed of 124 megabits per second. 124 megabits are 24 more megabits than the theoretically limit and look to be 28,5 megabits over the physical limit. This problem should mean that some clients will not get a stream, or all clients will lose many frames. In reality, for this measurement, all clients got the stream, and there were numerous frame drops. The rate of the stream lowered. The clients should have 60 FPS, but they instead got around 20 FPS. This problem means that one second of the stream played in three seconds on the clients. In the client's perspective, they stored frames in a small buffer. The frames were played on the screen together until the buffer was empty. Then the stream stopped until the small buffer was filled again. This might be the result of the tools such as Nginx, FFmpeg, and ffplay used to create Cedge. In this measurement of setup 7, the computers from Figure 5.2 were used as clients, and there was an additional five more computers that were used as clients. In this measurement, the clients stopped and started the stream at different times. It could almost look like the relay-node focused on filling a frame buffer for each client at a different time.

In setup 7, the average upload speed was also the highest upload speed with 92,5 megabits. The average download speed was also the highest so far, with 4,54 megabits. The RAM utilization increased with 19 megabytes to a total of 144 megabytes. That is the largest increase of RAM utilization in this experiment. However the CPU utilization decreased with 0,05%.

One remark for these setups in 6 and 7 and that is the 4k stream itself in worse quality. The quality was worse than the full HD or the 4k video before it was encoded. The 4k stream was still in 4k resolution. However, the frames looked more "grained" and with fewer updates to the frames during heavy motion frames. This might be a result of how the encoder encoded the video, or how the video itself was created.

This setup 8 is similar to setup 5, except with simulated WAN. This setup and the setup 5 have 15 clients each. The measurements would be expected to have a higher latency, and maybe a higher start-up time for the stream. The latency was not as expected. The seventh row in Table 5.2 shows the latency was lower than compared to the last measurement in the sixth row. This is very strange, the latency should at least have the latency from the last measurement with an additional 22 - 32 milliseconds. This measurement does also has the lowest amount of latency of all the measurements in Table 5.2.

There is a potential explanation for the low latency. The reason is that this measurement was done on another point in time than the other measurements. The relay-node was also rebooted right before it was measured. This could be the reason for the low latency. The network between the relay-node, source device, and the clients could be under less load now than when the other measurements where measured. The RAM utilization is also the lowest compared to the other results in Table 5.2, except the first row. The reboot of this relay-node might also be the reason for lower RAM utilization.

In Table 5.2 the hardware results of RAM and CPU does not have a clear logical pattern. Especially with the RAM utilization. This might be the result of numerous different aspects such as, other applications that suddenly use more RAM or CPU utilization. It might also be that the hardware components are very little utilized throughout the whole experiment, and combined with small spikes of other applications cause a significant variation for the measurement of the different setups. This unclear pattern could also be the result of when the measurements were taken in time. For example, if some of the measurements were taken around the five-minute timestamp, and some of the measurements were taken around the eight-minute timestamp. The frames around these two different timestamps in the stream could have very different scenes. One scene could be action-heavy with many large frame changes, and the other scene is slow-moving with few massive frame changes. The time column of Table 5.2 shows that most of the measurements were taken around the two-minute timestamp, and some around the three minute timestamp.

The streaming process seams to be utilizing more RAM when there are more clients connected to the relay-node. Except in row seven of Table 5.2 the RAM utilization is very low compared to the other tests. That test in row seven was retaken at a later time than the rest of the tests. The relay-node was rebooted for that test, while in the other tests, the relay-node was not rebooted. That could mean Nginx stores some data in RAM. The data could be logs, metadata, statistics, or some other temporary data. The data might be stored when Nginx has a longer uptime or is more accessed. It also looks like RAM is more utilized when the stream contains more data. Setup 6 in Table 5.2 shows a slightly increased RAM utilization with a higher data streams. Setup 7 also shows a larger increase of RAM utilization with higher data streams.

The CPU utilization has a slightly more logical pattern in Table 5.2. Setup 1-4 seams that the CPU utilization is dependant on the numbers of clients. However, the setups in 4-8 seams that is not the case. The only aspect that can be established with the CPU utilization is that it is in more use with Nginx. The relay-node was rebooted before the measurement in setup 8 of Table 5.2.

In the first experiment, the hardware utilization for RAM and CPU is not the dividing factor. The upload speed of the relay-node is primarily factor of this system. The bandwidth speeds could be fully utilized long before RAM, and CPU is fully utilized. However, this is good news for potential new relay-node devices. New relay-node devices only need a good Ethernet connection and lower hardware components.

Table 5.4: Setups for capabilities of Cedge

SETUP	RELAYS	CLIENT/RELAY	VIDEO	TC
9	1	1	HD	-
10	2	1	HD	-
11	3	1	HD	-
12	4	1	HD	-
13	5	1	HD	-
14	5	1	4K	-
15	5	1	HD	TC
16	5	1	4K	TC
17	5	3	4K	TC

5.3 Capabilities of Cedge

The second experiment will be about the actual system Cedge that is described in Chapter 3. The Cedge system will use what is described to be implemented from Chapter 4. This experiment consists of different measurements for the different setups of the system. The different setups of the system are described in user stories as this system uses a cooperative network. Where the users/-clients could be profoundly affected by the system, it is crucial to see that the improvement of a live video-streaming service does not come at the price of hijacking the bandwidth or edge device to the client.

The setup column of Table 5.4 is referred to for the different setups of measurements in this experiment. The results of the hardware for the different setups are displayed in Table 5.6, and the results of the latencies are displayed in Table 5.5.

The cooperative part of Cedge will use the client's resources to help distribute the stream. For the users' perspective, the Cedge system should use resources as little as possible if Cedge utilize too much of the hardware components as RAM and CPU. Then that could cause a problem for the edge device where the cooperative part of Cedge executes on. There could also be other applications executing on the edge device. This again, could be annoying for the clients. Another problem for the clients is if Cedge used too much bandwidth. The high utilization of bandwidth could cause internal problems, or reduces speed on the LAN. For example, if the user tries to send over a large file from one device to another on their home network. Then if Cedge utilizes 50% of the LAN, it could then take the client a longer time to send that file. The high utilization could also cause reduced internet connectivity to the WAN. In a similar example, where the client sends a large file to a receiver over the WAN. If Cedge use 50% of the upload speed, then the file transmitting could take

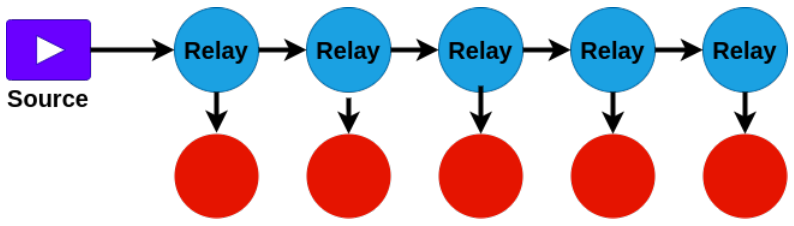

Figure 5.5: The test architecture

twice as long.

For the content distributors' perspective, Cedge should use the client's resources as much as possible. If the content distributor can use a large number of the client's resources, it could mean lower hardware costs for the distributor, which could also propagate down to the clients of a cheaper service. For example, if the content distributor has 1000 clients connected to one of their Content Distribution Networks (CDNs). In another potential system without relay-nodes, then that CDN needs to push the LVS out to 1000 clients. If 500 clients could receive the stream and cast it to another client, then the CDN could reduce the upload speed by almost 50%. The reason for almost 50% is that it would still need some extra data from all clients to coordinate the system. If the clients in an area have outstanding bandwidth and an excellent edge device, then maybe 50 clients could receive the stream from the CDN and cast it to 20 clients each. This could then lead to a reduction of almost 95% of the upload speed of the CDN.

The main reason for the setups in the second experiment is then to find this "golden balance". The golden balance of how Cedge could maximize cooperative clients, and how much of the clients' resources should Cedge utilize. The results for this experiment should hopefully give an estimate of where that balance should be.

First measurement

The first setup in this experiment is setup 9. It uses one source machine, five relay-nodes, and five clients. The source machine will push a LVS to a relay-node. The first relay-node will push that stream to the next relay-node, and the next after that relay-node. The Figure 5.5 shows an overview of this architecture of this experiment. The LVS will be in total pushed through five relay-nodes where the last relay-node in that chain will only get the stream and not push it to another relay-node. Each relay-node needs to serve at least one client with a stream.

This experiment has five setups, with an increasing number of resources. Each setup of this experiment will be measured separately. Setup 9 of the second experiment will start with only one relay-node and a client. The relay-node gets the stream from the source device and push it to the client. The relay-node will not push the stream to another relay-node in this part of the experiment. The hardware components of the relay-node are also measured. The components that are measured are CPU, RAM and network bandwidth.

In setup 10, the resources for this measurement have increased to two relay-nodes and two clients. It will work as setup 9, but this time the first relay-node will push the LVS to the second relay-node. This relay-node "chain" now contains two nodes in the chain. The second relay-node will not push the LVS to another relay-node. The hardware component of the first relay-node will also be measured.

Setup 11 will be very similar to setup 10. When the relay-node increases with one, it would also need an extra client. Which will add a node to the relay-node "chain". The hardware components of the first relay-node are measured for each relay-node that joined to the chain. In the end, the whole relay-node chain and the dedicated clients will look like Figure 5.5.

Setup 9-13

This experiment is where the Cedge system is measured. Setup 9 of this experiment is with one relay-node, a source device, and a client. The results for the hardware utilization of this measurement is in row one of Table 5.6, and the results for latency is row one in Table 5.5. The latency between the source device and the client of the relay-node is 616 milliseconds.

The next measurements of the setups in this experiment are setups 10-13 in Table 5.6 for the hardware results. The results for latency of setup 10-13 is in Table 5.5. The hardware results for these setups are very similar, except for the first measurement in setup 9. Setup 9 has only around half of the average upload speed as the other measurements. However, setup 9 only uploaded the stream to one client while the rest of the setups uploaded to both one client and another relay-node. So the lower average upload speed is anticipated in the measurement of setup 9. Setup 9 has 2,82 megabits in peak upload speed. Setup 12 has the highest average upload speed of 5,9 megabits.

Setup 9 has the lowest peak upload speed. Setup 9 also has around half of peak upload speed as the other setups, and it is the same reason for the low peak upload speed as the average upload speed in this setup. Setup 10 does have the highest peak upload speed of 8,86 megabits. Setup 13 has the lowest download speed of 2,52 megabits, and setup 12 has the highest average download speed of 3,06 megabits.

The RAM utilization is very similar for all setups in this experiment, except for setup 10. The reason for the low RAM utilization of setup 10 is that it was rebooted before the measurement of setup 10. The other setups were not rebooted before the measurement of that setup, and measurement of setup 10 was measured on a different time than the other setups. For the other setups, the RAM fluctuates between 126 megabits in setup 11 and up to 128 megabits in setup 9. The CPU utilization is very similar to all setups, even in setup 10, that was rebooted before the measurement of that setup. Setup 9 has the smallest measured CPU utilization with 0,15%. Setup 12 has the highest measured CPU utilization of 0,27%.

The results from latency in these setups are visualized in the Figure 5.6. This chart does only display "latency s-fc" in setup 9 since it is only one relay-node in that setup. So there cannot be measured an average latency and the "latency s-lf" in setup 9. The latency from the source device to the client of the first relay-node does fluctuate from 420 milliseconds in setup 13, and up to 763 milliseconds in setup 12. The latency from the source device to the client of the last relay-node increases for each extra relay-node. The latency from source to the client of the last relay-node start at 612 milliseconds in setup 10, and goes up to 5070 milliseconds in setup 13. It could also be interesting to see the average latency between each relay-node. The latency between the source and the client of the last relay-node is divided on the number of relay-nodes for each setup. This calculation should give an estimate of the average latency between relay-nodes. However, this is only an estimate of the average latency, and there could be a significant deviation of the latency between the relay-nodes in that particular setup. The average latency between each relay-node does increase from the setup 10-13. The average latency starts with 306 milliseconds in setup 10, and end with 1014 milliseconds. This increase of the average latency could be seen on the Figure 5.6.

The start-up time of a stream is measured with a client connected to the last relay node in the chain. The start-up time is displayed in Table 5.7. The start-up time for a stream to start playing on a client does fluctuate. Setup 9 has the lowest measured start-up time in this experiment of 4829 milliseconds. Setup 12 has the highest measured start-up time in this experiment of 7396 milliseconds.

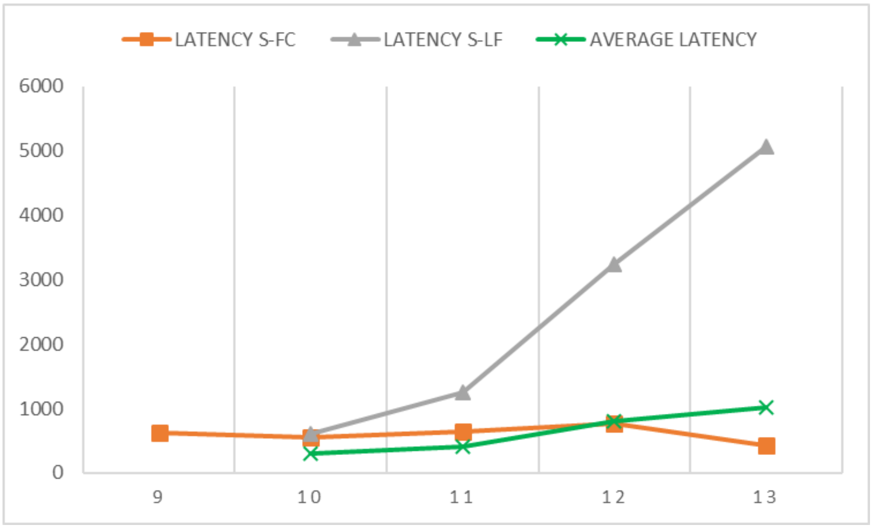

Figure 5.6: Latency for setups 9-13. The Y-axis is in milliseconds

Setup 14

Setup 14 is to compare the last measurements with more load on the system. There will be five relay-nodes that are chained together. The first relay-node receives the stream from the source device, and send it to the next relay-node in the chain. The difference in this setup compared to the other setups in this experiment is the stream. The stream is switched from a 30 FPS full HD to a 60 FPS 4k stream. The 4k stream should in theory, use eight times more data than the full HD stream.

The hardware result for this measurement of setup 14 is Table 5.6, and the results for latency is in Table 5.5. The latency from the source device to the client of the first relay-node 478 milliseconds. The latency from the source device to the client of the last relay-node 3418 milliseconds. The average latency for each relay-nodes is 683 milliseconds.

Setup 15

Setup 15 will also be about testing the actual system Cedge in the real world. This test will be very similar to setup 8. This test will also change the TC on the network card of the relay-nodes. Then the system could simulate a real network connection over the WAN while still being on the LAN. What is different in this test compared to setup 1 is that there are five relay-nodes. However, this test will have the same TC settings as the other TC test in setup 8. The relay-nodes will only have one client each. The new TC settings will only affect the connections between the five relay-nodes and the source device. The new settings will not directly affect the clients as they are on the LAN. However, the clients could be indirectly affected by the new settings in the relay-nodes. The network problems that arise in the relay-nodes might propagate down to the clients. It will be interesting to see how the clients could be affected by the change. Especially the difference between the client of the first relay node and a client of the last relay-node.

The latency result for setup 15 is displayed in Table 5.5, and the hardware results in Table 5.6. The latency from the source device to the client of the first relay-node is 593 milliseconds. The latency from the source device to the client of the last relay-node is 2823 milliseconds. The average latency for each relay-node is 564 milliseconds.

Setup 16

The setup 16 will combine both setup 14 and 15. The setup 16 will use the 4k stream, and also the TC changes to the network card to simulate a WAN on the LAN. In this setup, there will be five relay-nodes, where each relay-node will have one client each. Each relay-node will also have changes to the network card, and it will only directly affect the relay-nodes and the source device. The clients could then be indirectly affected by the TC changes to the network card.

The hardware result of the measurement in setup 16 is displayed in Table 5.6, and the latency is in Table 5.5. The latency from the source device to the client of the first relay-node is 920 milliseconds. The latency between the source device and the client of the last relay-node is 4050 milliseconds. The average latency for each relay-node is 810 milliseconds.

Setup 17

Setup 17 compares the last measurement from setup 16 with more load on the system. There will be five relay-nodes that are chained together. The first relay-node receives the stream from the source device. Each relay-node also serves three different clients the stream. The architecture of this will almost look like Figure 5.5, but each relay-node would have two more clients. The actual test looks more like the Figure 5.2, where the first row of clients connects to the first relay-node. The second row of clients are connected to the second relay-node, and so on. The hardware components of the first relay-node and the latency will be measured.

Table 5.5: Latency results of Cedge

SETUP	TIME	LATENCY S-FC	LATENCY S-LC	LATENCY FC-LC
9	339	0,616	-	-
10	310	0,552	0,612	0,06
11	318	0,636	1,246	0,61
12	380	0,763	3,237	2,51
13	450	0,42	5,07	4,65
14	351	0,478	3,418	2,94
15	449	0,593	2,823	2,23
16	347	0,92	4,05	3,13
17	354	0,466	2,586	2,12

Table 5.6: Hardware results of Cedge

TSETUP	RAM	CPU	PEAK	UP	DOWN
9	128MB	0,15%	4,53	2,82	2,85
10	117MB	0,18%	8,86	5,33	2,77
11	126MB	0,17%	8,29	5,05	2,61
12	127MB	0,27%	8,37	5,90	3,06
13	127MB	0,14%	8,35	4,58	2,52
14	127MB	0,26%	8,32	6,49	3,35
15	127MB	0,72%	8,92	5,51	2,87
16	127MB	0,63%	8,30	6,32	3,26
17	127MB	0,55%	16,7	12,8	3,44

The hardware result for this measurement of setup 17 is displayed in Table 5.6, and the latency is in Table 5.5. The latency from the source device to the client of the first relay-node is 466 milliseconds. The latency between the source device and the client of the first relay-node is 2586 milliseconds. The average latency for each relay-node is 517 milliseconds.

Table 5.5 and Table 5.6 show the results of the second experiment. The first column is very similar to the first column in Table 5.2. The difference between those two first columns is that this column will also have combined notions in parentheses. This first column will also have an extra notion called "C3". The "C3" notion describes that in this test, each relay-nodes have three clients instead of one. The second column in this table is equal to the second column of Table 5.2.

The third column is named "LATENCY S-FC", and this stands for "latency from the source device to the client of the first relay-node". This column in itself is very similar to the "latency" column of Table 5.2. The "lat S-FC" column is measured in seconds. The fourth column is named "LATENCY S-LC", and this stands for "latency from the source device to the client of the last relay-node". This fourth column is very similar to the third column. The difference is that it will instead measure the latency from the source device, through two or more relay-nodes, and then to the client of the last relay-node. The fifth column is named "LATENCY FC-LC". This column stands for "latency between the client of the first relay-node and the client of the last relay-node". This column shows how much longer the client of the last relay-node has to wait than the client of the first relay-node. The last five columns of Table 5.5 is equal to the last five columns of Table 5.2.

The Table 5.7 shows the result of the start-up time for a stream. The start-up time is the time it takes for a client to send a request to the relay-node, and when the LVS starts playing on the client's device. The time is shown in the second column named start-time, and it is measured in seconds. The first column in this table is named setup and refers to the setups in table Table 5.6.

5.3.1 Discussion

Setup 9 from this experiment and setup 1 from the first experiment have a very similar setup. The only difference is that setup 1 has three clients. While setup 9 has only one client. Both setups do also not push the stream to other relay-nodes. Only a few differences from the measurements of setup 9 and 1. The latency is 53 milliseconds longer setup 9. The RAM usage is also exceedingly increased from setup 1 to setup 9. Setup 1 use 119 megabits and setup 9 use 128 megabits. However, this could be because of the longer uptime of the relay-node. Where the relay-node had saved more data in RAM that is not relevant for these tests. CPU utilization is very similar. The bandwidth speeds are also very similar. The average upload speed is much lower in this measurement, but as anticipated since it has two fewer clients.

The hardware measurement from setup 14 looks to be very similar to the other setups. This was not anticipated for setup 14. In theory, the 4k stream in setup

Table 5.7: Start-up time results of Cedge

SETUP	START-TIME
9	4,829
10	4,660
11	5,002
12	7,396
13	5,623

14 should use eight times more data than the full HD stream in setup 9-13. So the average upload speed should, in theory, go from 8,35 megabits to 66,8 megabits. Thou, the average upload speed, is a noticeable amount higher than the other setups 9-13. However, the average upload speed is only around 0,6 megabits higher than the highest average upload speed from setup 9-13. The hardware utilization of RAM, CPU is also very similar as in setups 9-13.

In Table 5.6 the hardware results seams to be much more stable than the setups 1-5 in experiment one. The hardware utilization of RAM is very stable at around 127 megabits, except for setup 10. Setup 10 was the setup that was done at another time than the rest of the setups. The results of CPU utilization are also very stable. The CPU utilization fluctuates from 0,15% in setup 9 up to 0,72% in setup 15. The setup 10 is also very similar to the other setups despite its being rebooted.

5.4 Discussion

The actual measurements of the different setups for both experiments do have a small error margin. This error margin exists because of how the measurements were taken. The error margin is a latency problem. This problem could cause more latency in the results. However, the additional latency is only 2-5 milliseconds. This small amount of latency will not be considered for the rest of the evaluation.

The start-up times for the streams in setup 1-5 could be seen in Table 5.3. The start-up time for these setups is not in linear growth. Setup 1-5 does fluctuate from 3,804 seconds in setup 5 up to 4,373 seconds in setup 3. This could then be seen that the start-up time is not determined by the number of clients that are connected to the relay-node. There was a small expectation that there would be some fluctuating between different setups. The reason for the expected fluctuating results could be the size of the change in the frames sent out from the source device. The average start-up time for setup 1-5 is 4,022 seconds.

The start-up times for the stream in setup 9-13 could be seen in Table 5.7. The start-up time for these setups is also not in linear growth. Setup 9-13 does fluctuate from 4,660 seconds in setup 10 up to 7,396 seconds in setup 12. The average start-up time for these streams is 5,502 seconds. This is a significant increase of around 1,5 seconds in the start-up time of the stream, compared to the average start-up time for setups 1-5. This new start-up time could point to more variables than only the I-frames, that could affect the start-up time.

There is an essential aspect of the latency and start-up tests that need to be mentioned. In both experiments, the latency and start-up time tests were done by adding one or two extra clients. That means for each setup in setup 1-17 there was one or two additional clients that was not mentioned in Table 5.2, Table 5.5. There is one extra client in setups that only use one relay-node. The setups with only one relay-node are 1-8 and setup 9. The setups, with more than one relay-node, had two extra clients. The setups with more than one relay-node are 10-17. These extra clients were only connected during the measurement of the latency and start-up time. These measurements with extra clients were taken after the hardware measurements, such as RAM, CPU, and bandwidth. Therefore the hardware measurements should not be affected by the extra clients.

In the evaluation of the basic principles of Cedge, some aspects need to be discussed. The first aspect is when, in the video of the stream, the setups where measured. This aspect is relevant for both experiments. The time column in Table 5.2 and Table 5.5 shows the timestamp in the LVS where the setups was measured. The setups might get many results that are dependant on time in the setups was measured. This might be the aspect of the measurements that gives the largest deviation of the results. All of the tests should, in reality, be taken on the exact equal time. That would give the tests higher correctness. The measurement from Table 5.2 for the tests in experiment one was taken very closely around two to three minutes. Except for the test with 15 clients and with the 4k stream, that measurement was taken around three and a half minutes. In the second experiment Table 5.5 and Table 5.6 the measurements was taken around five to seven minutes.

The second problem with this aspect is that the timestamp only tells where the latency was measured. The RAM and CPU utilization measurements was taken before the latency measurements. The upload peak speed, average upload, and download speed were also measured before the latency. The start-up time for a stream to start was measured after the latency measurements. All of the measurements took around one to two minutes to measure.

The second aspect is the time frame of the measurements. The measurements of CPU, upload peak speed, average download, and upload speed are measured on a time frame. The time frame also varies from different measurements. The CPU have a time frame of 5 minutes, and there was also the option to use the one minute and the 15 minutes time frame. The five-minute time frame was a better version since it was very stable and did not change a lot like the one minute version. The 15-minute version is too large since the LVS only lasts for ten minutes. Measurement of only the Nginx CPU utilization would give better correctness of the tests. The measurements should also use a dynamic or self specified time frame for the average CPU load. This to specify the exact time the of when the measurements start.

The third aspect is the reboot of the relay-node. The relay-node should have been rebooted for each setup to give more precise measurements. This has been mentioned before in the evaluation. The main reason for rebooting is that over time RAM will especially fill up with temporary data. This data could take up a lot of the total RAM. At one point, the relay-node had an uptime for a month with minimal usage. The RAM utilization went from 110 megabytes at the start, to 230 megabytes at the end of the month. This large increase could give an insight into this particular uptime problem, combined with other applications that fill up the RAM. The result from setup 8 in Table 5.2 shows an large decrease in RAM. The measurement of setup 8 was done on another time than the rest of the setups in Table 5.2, and it was rebooted right before the test. Setup 10 in Table 5.6 was also done at another time than the rest of the tests. That test does also has a large decrease in RAM compared to the rest of the measurements of the setups in Table 5.6.

Another aspect is the LVS itself. The results of the setups might have an extra deviation because of the LVS itself. The first problem is that the video is pre-encoded. Pre-encoding removes some of the latency from the stream. Instead, the video should be live encoded to preserve some of the additional latency, and give a more correct and comparative result. However, this was done because of the massive amount of computer power the video encoder used. To give a better perspective of the time it took to pre-encode the two LVS that was used in the evaluation. It took one and a half hours for the full HD 30 FPS stream to be encoded. The 4k with 60 FPS version took around six hours to be encoded. The first attempts of LVS was with live encoding in the experiments. The live encoder used 100% of the CPU to encoded the video. If the encoder could not keep up with the stream speed, it would reduce the speed of the LVS. The problem with that is the clients suddenly got only half of the FPS. However, this problem could be solved with a more powerful CPU. Another reason for using pre-encoding is to remove unnecessary variables to give a better measurement of the fundamental aspects of the thesis itself. The second problem was that the video is also pre-encoded with the slowest speed to the compression ratio. The used quality is named "placebo", and this was not a recommended speed to compression ratio [49]. It is not recommended since it uses a significant amount of time to compress. The lower speed to compression ration means that it could send a better quality of the stream with less data. This means that it could send a higher quality LVS while also using less bandwidth. However, "placebo" was only about 1% better in quality than the next recommended setting [49].

6 Conclusion

The thesis stated that a cooperative live-streaming application could reduce the Total Cost of Ownership (TCO) of the streaming service provider, and reduce the bandwidth usage on the Internet as a whole. The live-streaming application incorporates edge devices in the Local Area Network (LAN) of clients, that distribute the live-stream through an overlay network.

The design of Cedge is built up with four layers. Where the live-stream starts its travel in the first layer. The live-stream is sent from a source device that is connected to the live-streaming application. The live-stream is sent to the second layer, which is an origin server that receives the stream. The origin server needs the help of a second component, the orchestrator, which organizes the edge nodes to the clients for the third layer. The first edge node in the third layer receives the live-stream from the origin server and distributes it to other edge nodes. Then the edge nodes will send the stream down to the fourth layer, which is the clients itself.

The source device, edge nodes, and the clients were implemented in this thesis for experimenting on the central aspect of the thesis statement. The experiments were divided into two categories of experiments. The first category used experimental setups to test the capabilities of an edge device. The second category used experimental setups to test the capabilities of the implementation of Cedge.

The results of the experiments show that the thesis is correct. Cedge shows that it can reduce the bandwidth from the content providers, and has the potential to reduce the bandwidth with several orders of magnitude. There is tremendous potential with edge devices in live video-streaming, as they are bandwidth limited and do not use a lot of other resources on the edge device. The edge devices also indicate that there are no problems with sending a live-stream through five edge nodes. However, there should be a limit to the number of edge nodes that a live-stream could travel through, as the latency increases with more nodes.

6.0.1 Future work

There are four aspects of Cedge that will be worked on in the future. One of these aspects is the orchestrator component of Cedge as it is needed to make effective organizing in the edge node substructures. The orchestrator has many possibilities to use many different attributes to organize the edge nodes. The orchestrator needs to divide the edge nodes into cooperative groups. The clients in the groups should be close to each other. The orchestrator could use Geo-IP to estimate the geographical area of an edge node, and group those edge nodes together. The orchestrator could also fine-grade the groups by measuring the latency between all edge nodes in an area. The orchestrator could further organize the groups by measuring their bandwidth to see if they should be the first or last edge node to get the stream.

The next aspect is that Cedge is only currently created for non-enterprise networks. As the enterprise network could have multiple LAN that is dispersed through a large area, or in many different geographical locations. The organizer would need to be able to take this into account when it is organizing the edge nodes. This could also create a problem for the edge node itself because now a single edge node might try to distribute a live video-stream too a whole university campus as an example. The edge node might need to create its own type of substructure inside a LAN of the enterprise network.

This thesis recommends having the edge node integrated in the home router, and there should have been a more straightforward way to install such an application. Therefore, the next aspect that will be worked on in the future is to integrate Cedge into some package-manager that could be easily installed and set up the router application. A type of package-manager does exist. There is a project called OpenWrt [59] that targets Linux operating system in embedded devices. OpenWrt could potentially be used for distributing the Cedge application for edge devices.

The last aspect for the future work in this thesis is an additional feature to Cedge, and this feature could create a better Quality Of Service (QOS) for clients. The feature is to divide a single live video-stream up to N-substreams. Where each substream gets the frames from the whole stream divided on N. If N is equal to three and the Frames Per Second (FPS) of the whole stream is 30 FPS, then each substream would be ten FPS. These substreams could, for example, be sent to three relay-node chain substructures. Each chain gets one substream from the origin server, and two other substreams from the other chains. Then each node in the chain needs to stitch up the frames in the right order and could then display it for the user. The reason for this feature is when one client suddenly exits during the stream, and then the whole stream will not be lost for the other clients in that chain. Only one-third of the FPS should be lost in this example.

Bibliography

[1] "Live-streaming is more popular than ever." https://www.theverge.com/2020/3/18/21185114/twitch-youtube-livestreaming-streamelements-coronavirus-quarantine-viewership-numbers.

[2] R. J. Glass, L. M. Glass, W. E. Beyeler, and H. J. Min, "Targeted social distancing designs for pandemic influenza," *Emerging infectious diseases*, vol. 12, no. 11, p. 1671, 2006.

[3] "World health organization, covid-19: physical distancing." https://www.who.int/westernpacific/emergencies/covid-19/information/physical-distancing.

[4] "Netflix will reduce its network traffic by 25 percent." https://www.theverge.com/2020/3/19/21187078/netflix-europe-streaming-european-union-bit-rate-broadband-coronavirus.

[5] "Netflix and youtube reduce streaming quality." https://edition.cnn.com/2020/03/19/tech/netflix-internet-overload-eu/index.html.

[6] "Live video-stream stats." https://livestream.com/blog/62-must-know-stats-live-video-streaming.

[7] "Twitter streamed more than 1300 live events in q1 2018." https://marketing.twitter.com/na/en/collections/video-on-twitter.

[8] "The averages of concurrent users and channels." https://twitchtracker.com/statistics.

[9] "Cisco excpect video-streaming to use 80% of all internet traffic.." https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1853168.

[10] "Breaks a twitch world record for concurrent viewers on twitch." https://www.cnet.com/news/drake-twitch-ninja-record-fortnite-

video-games/.

[11] "Online lecture." https://news.berkeley.edu/2020/03/23/coronavirus-forces-hands-on-learning-to-go-online-and-hands-off/.

[12] D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, P. R. Young, and P. J. Denning, "Computing as a discipline," *Commun. ACM*, vol. 32, p. 9–23, Jan. 1989.

[13] "The corpore sano center at uit." https://site.uit.no/corporesano/.

[14] P. Halvorsen, S. Sægrov, A. Mortensen, D. K. C. Kristensen, A. Eichhorn, M. Stenhaug, S. Dahl, H. K. Stensland, V. R. Gaddam, C. Griwodz, and D. Johansen, "Bagadus: An integrated system for arena sports analytics: A soccer case study," in *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, (New York, NY, USA), p. 48–59, Association for Computing Machinery, 2013.

[15] D. Johansen, M. Stenhaug, R. B. Hansen, A. Christensen, and P.-M. Høgmo, "Muithu: Smaller footprint, potentially larger imprint," in *Seventh International Conference on Digital Information Management (ICDIM 2012)*, pp. 205–214, IEEE, 2012.

[16] R. van Renesse, H. Johansen, N. Naigaonkar, and D. Johansen, "Secure abstraction with code capabilities," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 542–546, IEEE, 2013.

[17] H. D. Johansen, R. V. Renesse, Y. Vigfusson, and D. Johansen, "Fireflies: A secure and scalable membership and gossip service," *ACM Trans. Comput. Syst.*, vol. 33, May 2015.

[18] D. Johansen, P. Halvorsen, H. Johansen, H. Riiser, C. Gurrin, B. Olstad, C. Griwodz, Å. Kvalnes, J. Hurley, and T. Kupka, "Search-based composition, streaming and playback of video archive content," *Multimedia Tools and Applications*, vol. 61, no. 2, pp. 419–445, 2012.

[19] R. W. M. Lee, "Content delivery network system and method for network configuring," Oct. 26 2004. US Patent 6,810,417.

[20] J. E. Fowler and R. Yagel, "Lossless compression of volume data," in *Proceedings of the 1994 Symposium on Volume Visualization*, VVS '94, (New York, NY, USA), p. 43–50, Association for Computing Machinery, 1994.

[21] "H264 codec." https://www.itu.int/rec/T-REC-H.264.

[22] "High efficiency video coding (hevc)." https://www.itu.int/rec/T-REC-H.265.

[23] "Vp9 codec." https://www.webmproject.org/vp9/.

[24] "Av1 codec." https://aomediacodec.github.io/av1-spec/av1-spec.pdf.

[25] "Hevc patent portofolio license briefing." https://web.archive.org/web/20141006091331/http://www.mpegla.com/main/programs/HEVC/Documents/HEVCweb.pdf.

[26] "H264 patent portofolio license briefing." https://www.mpegla.com/wp-content/uploads/avcweb.pdf.

[27] M. Karczewicz and R. Kurceren, "The sp-and si-frames design for h. 264/avc," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 637–644, 2003.

[28] "Ipb-frames." https://www.canon.com.hk/cpx/public/images/tech/article/video/EOS_Movie_Compression_Options_All_I_and_IPB/IPB_Compression_EN_s.jpg.

[29] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974.

[30] L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, and D. Stodolsky, "A transport layer for live streaming in a content delivery network," *Proceedings of the IEEE*, vol. 92, no. 9, pp. 1408–1419, 2004.

[31] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang, "Practical, real-time centralized control for cdn-based live video delivery," *SIGCOMM Comput. Commun. Rev.*, vol. 45, p. 311–324, Aug. 2015.

[32] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang, "Practical, real-time centralized control for cdn-based live video delivery," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 311–324, 2015.

[33] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky," in *Proceedings of the 17th ACM international*

conference on Multimedia, pp. 25–34, 2009.

[34] S. Seyyedi and B. Akbari, “Hybrid cdn-p2p architectures for live video streaming: Comparative study of connected and unconnected meshes,” in *2011 International Symposium on Computer Networks and Distributed Systems (CNDS)*, pp. 175–180, IEEE, 2011.

[35] M. Zhang, J.-G. Luo, L. Zhao, and S.-Q. Yang, “A peer-to-peer network for live media streaming using a push-pull approach,” in *Proceedings of the 13th Annual ACM International Conference on Multimedia*, MULTIMEDIA ’05, (New York, NY, USA), p. 287–290, Association for Computing Machinery, 2005.

[36] J. He, A. Chaintreau, and C. Diot, “A performance evaluation of scalable live video streaming with nano data centers,” *Computer networks*, vol. 53, no. 2, pp. 153–167, 2009.

[37] “Total fiber connection to norwegian households.” `https://ekomstatistikken.nkom.no/#/statistics/details?servicearea=Fast%20bredb%C3%A5nd&label=Fast%20bredb%C3%A5nd%20-%20abonnement`.

[38] “The largest broadband companies in norway.” `https://ekomstatistikken.nkom.no/#/statistics/details?servicearea=Fast%20bredb%C3%A5nd&label=Fast%20bredb%C3%A5nd%20-%20st%C3%B8rste%20tilbyder%20(abonnement)`.

[39] “Telenor fiber broadband speeed subscriptions.” `https://www.telenor.no/privat/internett/fiber/`.

[40] D. Chatzopoulou and M. Kokkodis, “Ip geolocation,” *Computer Science and Engineering Dept, UC Riverside, Tech. Rep*, 2007.

[41] “Nginx github.” `https://github.com/nginx/nginx`.

[42] “Rtmp module to nginx.” `https://github.com/arut/nginx-rtmp-module`.

[43] “Nginx configuration parameters.” `http://nginx.org/en/docs/ngx_core_module.html`.

[44] “Nginx rtmp module configuration parameters.” `https://github.com/arut/nginx-rtmp-module/wiki/Directives#rtmp`.

[45] “Nginx http module configuration parameters.” `http://nginx.org/en/docs/http/ngx_http_core_module.html#http`.

[46] "Ffmpeg tool for record, convert and stream audio and video." https://ffmpeg.org/.

[47] "Ffmpeg github." https://github.com/ffmpeg/ffmpeg.

[48] "Logitech article about encoding in-camera." https://www.logitech.com/en-us/video-collaboration/resources/think-tank/articles/article-logitech-and-h264-encoding.html.

[49] "h264 encoding preset." https://trac.ffmpeg.org/wiki/Encode/H.264.

[50] "Ffmpeg flags for streaming." https://trac.ffmpeg.org/wiki/EncodingForStreamingSites.

[51] "Ffmpeg flags." https://ffmpeg.org/ffmpeg.html.

[52] Y. Liu, D. Niu, and B. Li, "Delay-optimized video traffic routing in software-defined interdatacenter networks," *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 865–878, 2016.

[53] R. B. Friedman, S. M. Parekh, and B. Lutch, "Geo-intelligent traffic manager," Nov. 30 2010. US Patent 7,844,729.

[54] Z.-Y. Zhuang and P. Zhou, "Optimized selection of streaming servers with geodns for cdn-delivered live streaming," 路技刊, vol. 15, no. 2, pp. 285–296, 2014.

[55] B. Tullemans, "Technical practices for a multi-cdn distribution strategy," *SMPTE Motion Imaging Journal*, vol. 127, no. 6, pp. 1–7, 2018.

[56] V. Khare and B. Zhang, "Making cdn and isp routings symbiotic," in *2011 31st International Conference on Distributed Computing Systems*, pp. 869–878, IEEE, 2011.

[57] "Big buck bunny." http://bbb3d.renderfarming.net/download.html.

[58] "Akamai state of the internet." https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-report.pdf.

[59] "Linux operating system for embedded devices." https://openwrt.org/.