Cooperative Peer-to-Peer Video Clients

John O Sullivan

Final-Year-Project Bsc in Computer Science

Supervised by: Dr. Jason Quinlan

Department of Computer Science

University College Cork

April 2020

Abstract

Video streaming is the most in demand form of entertainment on the internet today. Whether it is video on-demand services such as Netflix or short videos on your Facebook feed, video is the dominant form of online media. The 2020 Mobile Internet Phenomena Report shows that mobile video downstream traffic accounts for more than 65% of all mobile internet traffic, a massive 23% increase in comparison to the 2019 report.[1] This growth in video consumption will continue for years to come. When this is combined with the introduction of the Internet of Things and the growing popularity of high resolution video, it is easy see how bandwidth is becoming a precious commodity. This project walks through the design and successful implementation of a Cooperative Peer-to-Peer video streaming network. This Cooperative network will provide a system in which peers can share previously downloaded video content over the local network. Peers will then be able to use the Cooperative network to search for and stream video content without any request to a web server reducing the outgoing traffic on the network. This Cooperative network is capable of reducing the number of outgoing web server requests by as much as 80% in some cases, while maintaining or even improving the quality of experience of video clients when compared to the quality of experience of a standard DASH video player. This projects success in implementing a Cooperative Peer-to-Peer streaming network hopes to prove the viability and effectiveness of a Cooperative network in alleviating the amount of bandwidth required to stream video in the internet today.

Declaration of originality

Declaration of Originality In signing this declaration, you are conforming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award. I hereby declare that:

- this is all my own work, unless clearly indicated otherwise, with full and proper accreditation;
- with respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;
- with respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Signed: John O Sullivan
Date: 16/04/2020

Contents

1 Introduction

This chapter gives an overview of the project structure and underlying technologies as well as setting out the goals of the project.

1.1 Growth of online Video

YouTube (an online video sharing platform where a user could upload and consume video) was launched in 2006. It quickly gained popularity and was bought by Google in 2007. YouTube's popularity and success was unprecedented, with some estimating in 2007 that YouTube consumed as much bandwidth (The rate of data transfer or throughput in an internet connection) as the entire internet in 2000 [2] . YouTube's popularity continued to explode and in 2010 YouTube had 1 billion active users. This massive user base meant YouTube was using an enormous amount of the Internet's bandwidth. YouTube was not the only online video platform seeing massive growth. In 2007 Netflix (an online payed subscription based service offering hundreds of movies and TV shows to watch at any time)launched its online streaming service. Netflix like YouTube saw an explosion in popularity. By 2011 Netflix accounted for 27.6% of all internet traffic in North America [3]. Video was quickly becoming the biggest source of traffic on the internet. In 2011 Netflix and YouTube combined accounted for 37.6% of all American internet traffic [3].

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	47.55%	Netflix	32.69%	Netflix	29.03%
2	HTTP	11.45%	HTTP	17.48%	HTTP	16.59%
3	Netflix	7.69%	YouTube	11.32%	BitTorrent	13.47%
4	Skype	4.27%	BitTorrent	7.62%	YouTube	9.90%
5	SSL	3.57%	Flash Video	3.41%	Flash Video	3.04%
6	Facebook	2.19%	RTMP	3.12%	RTMP	2.81%
7	PPStream	1.73%	iTunes	3.05%	iTunes	2.69%
8	YouTube	1.64%	Facebook	1.78%	SSL	1.96%
9	Xbox Live	1.31%	MPEG	1.72%	Facebook	1.84%
10	Teredo	1.25%	SSL	1.69%	MPEG	1.49%
	Top 10	**82.63%**	**Top 10**	**83.88%**	**Top 10**	**82.83%**

Figure 1: Top Peak Period Applications by Bytes (North America, Fixed Access)(Taken from the 2011 Global Internet Phenomena Report by Sandvine released Spring 2011) [3]

This demand for online video will continued to grow. With the update of HTML5 in 2015 came support for Dynamic Adaptive Streaming over HTTP or DASH [4]. DASH allows the video player to select higher or lower quality video content based on the quality of the Internet connection available. This helped to ensure video clients received the best quality content they could while also helping to reduce the bandwidth consumption of online video streaming platforms. DASH has been widely adopted and is now the default player for YouTube, Netflix and many more online video streaming platforms. A study analysing the impact of using the DASH player on YouTube found that it achieved bandwidth cost savings of 95% on low quality video content (up to 480p) and 83% bandwidth cost savings on HD video content (up to 1080p) [5]. Despite this greatly improved utilisation of bandwidth the popularity of video continued to grow. In 2019 video streaming accounted for 60% of the worlds internet consumption [6]. Of this 60%, Netflix, YouTube and HTTP Video Streams (Video streaming content not currently tracked individually such as cable or broadcast channels) account for 59.6% of all video content consumption [6]. Furthermore video now accounts for 65% of all mobile internet traffic [1].

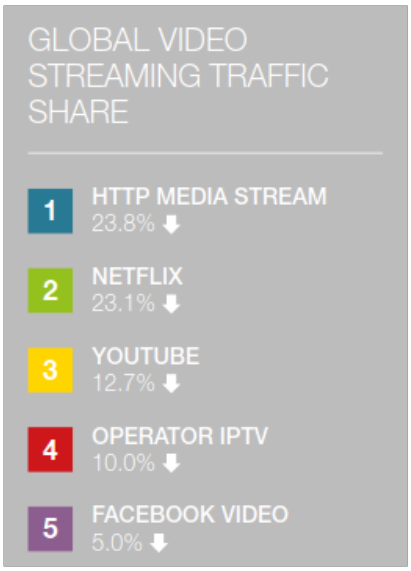


Figure 2: Top Peak Period Applications by Bytes (North America, Fixed Access)(Taken from the 2019 Global Internet Phenomena Report by Sandvine released September 2019) [6]

This enormous demand for video will only continue to grow with the recent introduction of 4k content, and the coming introduction of 8k content further adding to the vast amount of bandwidth used by video. With such a high proportion of today's internet capacity dedicated to video streaming it is easy to see bandwidth is becoming a precious commodity. The effects of this can be seen even today. During the current COVID 19 of 2020, a large proportion of people are being asked to work at home. This sudden demand for extra bandwidth coupled with a large amount of people being confined inside has led to enormous demands for bandwidth on Internet Service Providers or ISPs. In order to reduce the impact of such demand the EU has asked YouTube, Netflix and other video streaming platforms to reduce the quality of the video content they provide to alleviate the pressure on the ISPs[7]. This reduction in video quality has a negative impact on Quality of Experience (QOE) for users all around Europe. There are many QOE metrics available each focusing more heavily on a different aspect of the video streaming experience ranging from buffering times to maximum video quality. The goal of QOE metrics is to quantify how good of an experience the video streaming client has while consuming video content. This reduction in

quality of video content has quite a negative impact on the overall QOE.

The main issue regarding the internet and video streaming is that the internet was not designed for video. The internet was designed to deal with web pages and images. In comparison to video content web pages and video are relatively small files. Furthermore web pages and images are not sensitive to latency (Time taken for content to arrive at client once requested) allowing for a centralised model to work very effectively. Latency is an issue in a centralised model due to the great distance content must traverse to reach your device. Video content is extremely sensitive to latency making the centralised model not suited to hosting this form of media. In order to mitigate the effect of latency modern networks use Content Distribution Networks or CDNs [8]. A CDN consists of hundreds of cache servers placed in strategic geographical positions in order to minimise the distance from the server to the client. A central server pushes video content to the CDN's cache servers. These servers then act as the main video server. This moves the content significantly closer to the requesting client and greatly reduces the strain on the main server. Despite this great reduction in latency cache servers can still be hundreds of kilometers away from the requesting client. This means that in many cases there is still significant latency present in the network. However there are other ways alleviate this demand on ISPs. Peer-to-Peer networks [9] offer another solution. Peer-to-Peer networks offer a way for devices to share content among themselves without the need of a central web server. This helps to alleviate the primary bottleneck in today's Internet, the connection to the web server, while massively reducing latency due to the shorter distance that is traversed by content. This Peer-to-Peer model is what this project hopes to utilise and exploit. This project hopes to create a network that will allow video streaming clients in the same local area to share video content among themselves instead of relying on the web server. This model will track what content that is currently available on local Peers and allow Peers in the same network to request this video content from another Peer instead of from the web server. Allowing these Peers to share video content between themselves will help to alleviate the bandwidth needed to meet video demand. This would improve the internet for all users not just those streaming video content.

1.2 Project Goals

- Create a fully functioning Peer-to-Peer Cooperative Network capable of sharing video content over a local network
- Investigate current QOE on various network configurations using existing network model
- Compare and contrast cooperative network model performance to standard network model
- Provide evidence that a cooperative model can maintain or improve QOE for the end user while also reducing the load on the web server

1.3 Report Outline

Chapter 2 will outline the background of the project explaining the technologies utilised and design decisions taken in the creation of the Cooperative Streaming Model. Chapter 3 will outline the implementation and development of the Cooperative streaming model while chapters 4 and 5 will explain the evaluation and analysis of this project.

2 Background

This projects main goal is to implement a cooperative peer-to-peer video streaming network. This model will consist of a local server capable of tracking what content and peers are currently available on the network and a number of video streaming clients capable of requesting and sharing content to and from other video streaming clients. In this model when a video streaming client requests video content it first searches for this video content among its known neighboring clients. If the requested content is not available from its neighboring clients the requesting client will then query the local server. The local server tracks all video content and peers on the network. If the content is present on the local network the local server returns the address of the client hosting this content to the requesting client. The requesting client then requests and receives this content from the client hosting the requested content. If the content is not available on the local network the requesting client will request the content from the internet as normal.

To create this model the utilisation of many technologies was necessary. This chapter explains the key technologies utilised during the course of this project and their purpose in the cooperative peer-to-peer video streaming network.

2.1 Project Technologies

2.1.1 Peer-to-Peer Networks

Communication between clients is vital in the Cooperative Peer-to-Peer video streaming model. Communication between clients will allow them to share previously downloaded video segments alleviating the load on the central web server. In this project we will be utilising the Peer-to-Peer network model to allow for communication between clients. Peer-to-Peer networks are a staple in today's internet. Peer-to-Peer networks offer a way in which devices can share content without the need of a central server. Each Peer, or device, on the network acts as both server and client. This allows the sharing of content or resources, such as memory or processing power, with other Peers. Peer-to-Peer networks offer an opportunity to host and distribute video content without central servers. This eliminates a primary bottleneck in the network. Today there are many forms of Peer- to-Peer networks.

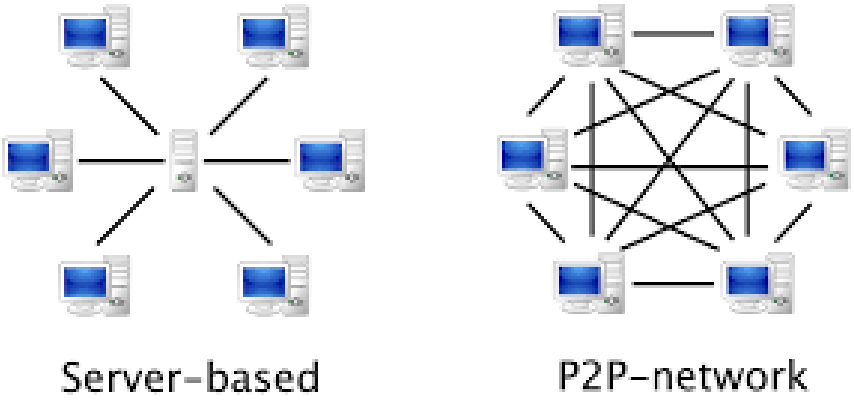


Figure 3: Example of difference between client server architecture and peer-to-peer architecture(Source: https://www.pinterest.ie/pin/457185799650805573/)

Unstructured Peer-to-Peer networks do not organise the nodes connected the the network. When searching the network peers communicate randomly with one another. These unstructured systems are highly robust against churn (several peers joining or leaving the network at a time). Although easier to build unstructured networks they are more resource intensive needing more CPU power and memory to search the network as queries are based on reaching as many peers as possible. This tends to flood the network with queries particularly when content is not widely available.

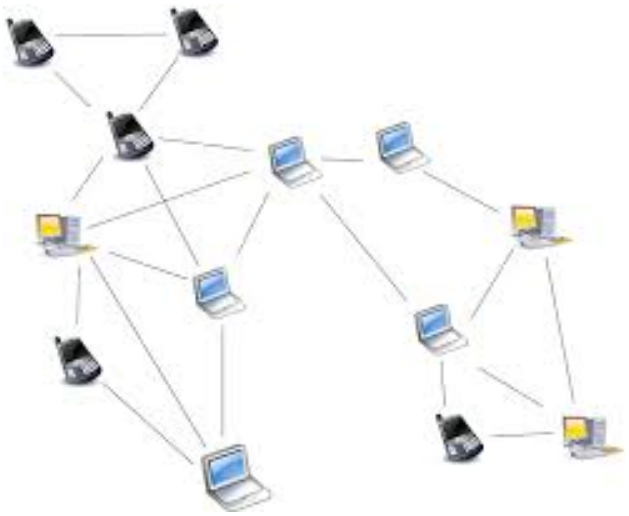


Figure 4: Example of unstructured peer-to-peer architecture(Source: https://en.wikipedia.org/wiki/Peer-to-peer

In comparison **Structured Peer-to-Peer** networks use some organisational architecture. This allows peers to efficiently search the network for content even in the case where content is not widely available. In most cases hash functions are used to facilitate database lookups. While structured networks are more efficient they have higher levels of centralisation (entire network depends on a small few peers) and maintenance costs. They are also less robust when faced with high levels of churn.

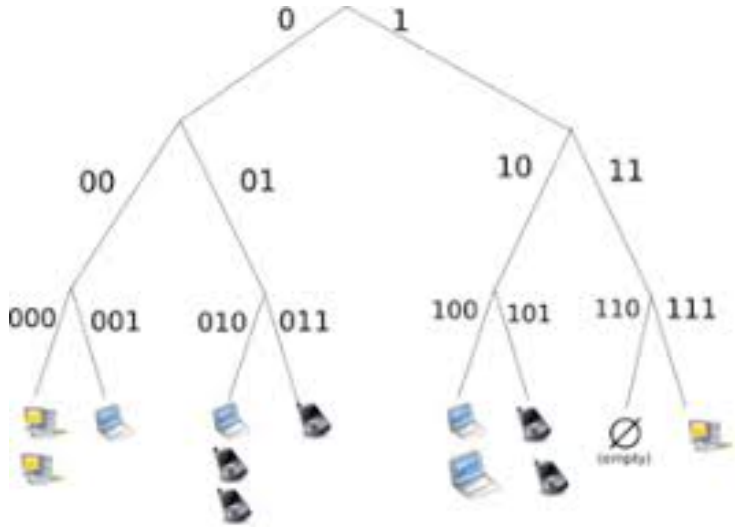


Figure 5: Example of structured peer-to-peer architecture(Source: https://en.wikipedia.org/wiki/Peer-to-peer

Both Structured and Unstructured peer-to-peer networks are considered true peer to peer networks as they do not have a central server. **Hybrid Peer-to-Peer** [9] networks combines the advantages of both a standard client server architecture and peer-to-peer systems. In this structure there is a central server. However, unlike traditional central servers, this server will not distribute any content. Instead it simply keeps track of the Peers and content available in the network. Peers can consult this server when joining the network or when searching for specific content that the peer cannot find amongst its own neighboring peers. Once a Peer has joined the network it can communicate with other Peers, as in a standard Peer-to-Peer network. Peers will rarely have to consult the central server after joining the network. Hybrid Peer-to-Peer systems generally offer improved performance over Structured and Unstructured networks but at the disadvantage of the centralisation of the local server and the cost to maintain the local server.

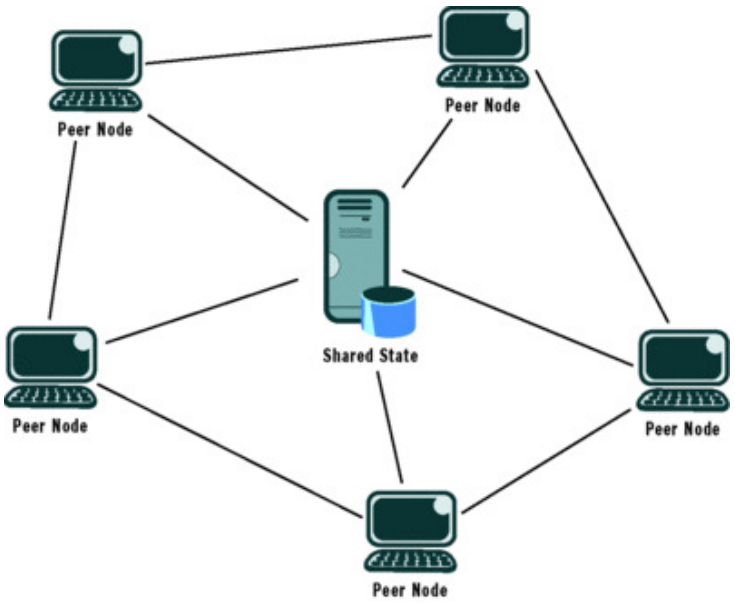


Figure 6: Example of hyrbrid peer-to-peer architecture(Source: https://docs.microsoft.com/en-us/archive/msdn-magazine/2008/july/wcf-p2p-how-to-design-state-sharing-in-a-peer-network

A true Peer-to-Peer network would be preferable as no central server would be needed. The lack of a central server means the project could be implemented on existing hardware with no changes. Despite its advantages a true Peer-to-Peer network is out- side the scope of this project due to the complex organisation algorithms and costly maintenance associated with them. This complexity coupled with current time constraints makes a true Peer-to-Peer implementation unattainable which is why a Hybrid Peer-to-Peer network was chosen for this project.

2.1.2 Protocol Buffer

In order for information to be passed between peers efficiently the data being passed must be serialised (translating data to a format that is quicker to send that can be reconstructed once received). Serialisation is vital in order to create an efficient and fast network. Protocol Buffers [10] are an automated mechanism for serialising structured data similar to Extensible Markup Language (XML). In contrast to XML Protocol Buffers are smaller and far more simplistic to create and manage. To use Protocol Buffers you

must first specify how you want the information that is to be serialised structured. This structure is specified in a .proto file. Every Protocol Buffer is a small logical record containing a series of name-value pairs.

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

Figure 7: Example of .proto file for generating Protocol Buffer data access classes(Source: https://developers.google.com/protocol-buffers/docs/overview)

Once you have defined your Protocol Buffer messages in a .proto file you must then use the protocol buffer compiler for your relevant language to generate data access classes. You can then utilise these classes in your application. Some more advantages of using Protocol Buffers over XML are as follows:

- 3 to 10 times smaller
- 20 to 100 times faster
- less ambiguous
- easy to use data access classes
- easy port across programming languages
- allows for modification of the protocol while not breaking existing code

For the above reasons Protocol Buffers are used in this project.

2.1.3 General-Purpose Remote Procedure Calls(gRPC)

In order for the peers to send and receive messages each peer must have an Application Programming Interface(API) allowing peers to quickly and efficiently send and receive data over the network. gRPC is an open source high performance Remote Procedure Call (RPC) framework. gRPC is based on the HTTP2 standard. gRPC enables the easy creation of Application Programming Interfaces. These API's are efficient and scalable and provide a wide range of features such as authentication, flow control, bi-directional streaming, timeouts and cancellations. gRPC is CPU efficient offering low latency and low bandwidth usage making it perfect for implementing a lightweight Hybrid Peer-to-Peer Network. gRPC utilises Protocol Buffers allowing for easy service definition and automatic generation of client libraries. gRPC has been adopted by a wide range of companies some of the most notable are Netflix, Cisco and Docker [11].

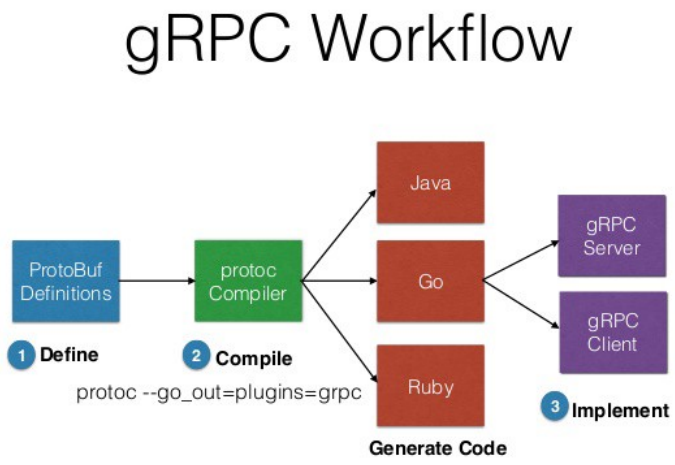


Figure 8: Example of gRPC workflow
`https://medium.com/@akshitjain_74512/inter-service-communication-with-grpc-d815a561e3a1`

Once the data structures are defined in the Protocol Buffers protoc (A protocol buffer compiler to generate gRPC code) is used to create the gen-

erated code for your desired language in this projects case that language is Golang. The generated code provides a gRPC server and gRPC client classes with the correct Protocol Buffer functions predefined. These classes can then be used to implement gRPC clients and servers that can communicate quickly and efficiently.

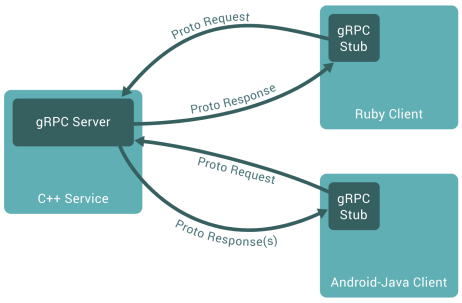


Figure 9: Example of gRPC network structure
`https://towardsdatascience.com/grpc-in-golang-bb40396eb8b1`

2.1.4 Consul Key-Value(KV)

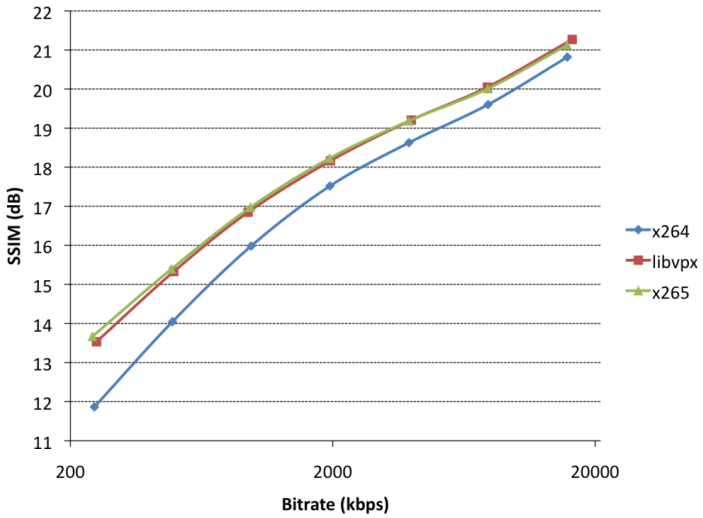
The cooperative peer-to-peer streaming model requires the use of a local server capable of tracking the video content and peers currently connected to the network. Consul is this projects chosen solution. Consul is a Service networking solution used to connect and secure services across any run time platform and public or private cloud. It offers a wide array of features including its Key-Value or KV API. Consul's KV system allows you to map a key(string) to a Value(byte) that can then be accessed or modified through Consul's HTTP API. Consul's KV system offers a convenient and efficient way to track what content is currently available on our cooperative network. This allows devices to quickly query the Consul server for the address of requested video content. Consul also offers the opportunity for multiple Consul servers to share information using a Gossip Protocol (Information is shared between neighbors). This allows this projects Cooperative Client model to scale more easily.

2.1.5 Video Encoding

To test the Cooperative Peer-to-Peer video streaming network the project needed video content suitable for video streaming. Video encoding is used to create suitable video content for streaming.

Video encoding is the process of converting raw video files to formats that are more suitable for transporting over the network and more suitable for devices to playback. Raw video files are too large to be processed and downloaded in an efficient manner. When it comes to streaming videos are often compressed from gigabytes of data to megabytes of data. To shrink video content to this more manageable size video codecs (coder-decoders) are used. Codecs shrink the video content for storage and transport but allow for later decompression to view the video content. There are many codecs available today.

H.264, also referred to as Advanced Video Coding (AVC), is a video codec. It is a standard based on block-oriented, motion compensated integer-DCT encoding. H.264 supports high resolution content up to 8K UHD [12] and as of September 2019 is used by 91% of video developers [13]. The goal of H.264 was to provide high quality video content with substantially lower bit rates than previous standards. This was to be done without increasing encoding complexity so much that it became impractical to use or too expensive to implement. This was achieved using many features such as integer DCT encoding, variable block size segmentation and multi-picture inter-picture prediction. H.264's encoding works by converting HDMI audiovisual signals to an IP stream that can be transmitted over an IP network. Once the IP stream is received it is converted back to an uncompressed HDMI format. Due to H.264's substantially lower bitrates (80% lower than Motion JPEG and estimated to be 50% lower than MPEG-2) and reduced storage demands it is commonly used in many application including Blu-ray Discs and many online streaming websites such as Netflix, YouTube, Hulu and Prime Video.



https://blogs.gnome.org/rbultje/2015/09/28/vp9-encodingdecoding-performance-vs-hevch-264/

Figure 10: Graph plotting SSIM(measurement of video quality) and bitrate of multiple codecs

H.265 is H.264's replacement. It offers greater compression in comparison to H.264 offering smaller file sizes. This makes it ideal for streaming and a perfect replacement for H.265. Unfortunately H.265s adoption has been hampered due to uncertainties surrounding the payment of royalties. As a result it is not widely used in industry.

VP9 is a codec standard developed by Google its compression is similar to H.265 but it offers greater quality content at a lower bitrate. Unfortunately similarly to H.265 it has not been widely adopted despite support for VP9 on YouTube and all Android devices.

While better performing codecs exist H.264s widespread adoption in today's internet makes it the most suitable codec to use in testing the Cooperative Peer-to-Peer streaming model as it has the most real world relevance.

2.1.6 Dynamic Adaptive Streaming over HTTP(DASH)

In this project each client must be capable of streaming video. Dynamic Adaptive streaming over HTTP (DASH) [4] is the chosen streaming method for this project. DASH was chosen as it is the most prominent streaming

method used in the internet today, with video streaming giants like YouTube and Netflix using DASH its popularity in industry is clear. DASH's popularity is why it is utilised in this project.

Before DASH, HTTP Adaptive Streaming (HAS) was the main streaming solution. The image below shows the structure of a HAS streaming system.

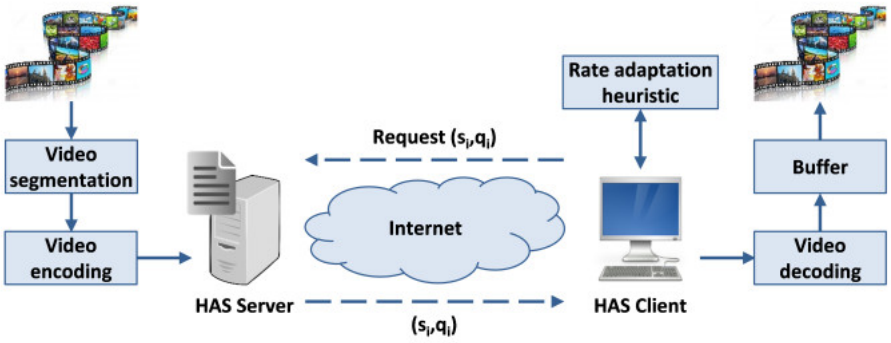


Figure 11: Example of HAS streaming network structure(Source: [14]

In HAS video content is first encoded in a wide variety of quality levels. Each level is determined by the average bitrate of the level. Furthermore video content is divided into segments typically of lengths in the range of one to ten seconds. Each segment can be downloaded and decoded independently. To initiate streaming the client first downloads the manifest file. The manifest file is a record of all segments and quality levels available for streaming. The Rate Determination Algorithm (RDA) is used to select the quality of the next segment to be downloaded. This selection is based on current network conditions and the buffer filling level. The objective of the RDA is to maximise the quality of the user experience when streaming by minimising buffering and maximising video quality.

DASH is the first adaptive bit-rate HTTP-based streaming solution that is an international standard. Dash makes use of existing HTTP web server infrastructure that is used for effectively all World Wide Web content. This use of existing infrastructure allowed it to be quickly adopted. Many prominent companies including Netflix and YouTube now use Dash as a replacement for slow running and insecure programs like Flash and Silverlight. Dash is codec agnostic meaning it can utilise content in any codec format. This project will be using the H.264 codec. This project focuses primarily on Video on

demand. H.264 is the prominent codec used by VOD services such as Netflix and YouTube. As a result it was most relevant to use H.264 as the encoding for this project.

2.1.7 goDASH

In this project each client must be capable of streaming DASH content. For this reason we are using goDASH. GoDASH [15] is a configurable headless DASH player implemented in Golang. GoDASH includes options for the use of many different adaptation algorithms (Arbiter,Bba2,Conventional,Elastic and Logistic), video codecs (h265,h264,AV1) and DASH profiles (full, main, live, full_byte_range and main_byte_range). GoDASH uses a modifiable configuration file. GoDASH also contains options for debug information, requests for MPD urls and per clip segment header information . Table 1 illustrates the options available for per segment log output. The default headers are always printed to logs while the optional headers can be configured to be "on" or "off" in the configuration file. GoDASH provides five qualitative estimation metrics (Qem). These QeM's provide a blend of the options currently available. These QeMs are calculated in real time making them available for offline evaluation at a later stage. The five QeMs used are as follows:

Duanmu[16] is a QOE measure given as the following equation.

$(2.3 * initDelay) - (56.5 * rebufferPercentage) + (0.0070 * avgBitrate) + (0.0007 * avgRateSwitching)$ Duanmu penalises buffering heavily while rewarding streaming with a higher bitrate. Duanmu is a good measure of how often and how significant the buffering of content in the Dash player is. Duanmu returns a in the range of [0,1] with lower values representing a better QOE.

Claey[17] is a QOE measure given by the following equation.

$0.17 + rateQOE - switchingPenalty - stallPenalty$ claey penalises high amount of bit rate switching(switchingPenalty). It also penalises for a stalling of content in the player (stallPenalty). It rewards a high average bitrate. (rateQOE) claey is a good measure of having a consistent image quality that does not change significantly or often. Claey returns a value in the rate of [0,50] with higher values yielding a better QOE.

Yu[18] is a QOE measure given by the following equation. $avgBitrate - switchingQOE - starvation$ Yu is similar to claey. IT penalises frequent bitrate changes and stalling of content. However in contrast to claey it places

a far greater importance on starvation penalising very heavily. It rewards a high average bitrate. This is a good indication of stalling impact on video quality. Yu returns values in the range of [0,5] with higher values representing a better QOE.

Type	**Description**
Default Output:	
Seg_#	Streamed segment number
Arr_Time	Arrival time in milliseconds(ms)
Del_Time	Time taken to deliver the segment (ms)
Stall_Dur	Stall duration
Rep_Level	Representation quality (kbps)
Del_Rate	Delivery rate (kbps)
Act_Rate	Actual rate (kbps)
Byte_Size	Byte size of this segment
Buffer_Level	Buffer level (ms)
Optional Output:	
Algorithm	Adaptive algorithm
Seg_Dur	Segment duration
Codec	Video encoder
Width	Representation width in pixels
Height	Representation height in pixels
FPS	Frame rate of the streamed video content
Play_Pos	Current Playback position
RTT	Packet level (ms)
Protocol	HTTP protocol
QeM Output:	
P.1203	P.1203 standard scale [0,5]
Clae	Clae Model scale [0,5]
Duanmu	Duanmu Model scale [0,100]
Yin	Yin model scale dependent on HAS bitrate
Yu	Yu model scale [0,5]

Table 1: Notation used in the goDASH output logs

Yin[19] is a QOE measure given by the following equation. $(sumsegRate/1000) - (1 * avgSwitchingMagnitude) - 3 * totalStall$ Yin rewards a high total bitrate count(The sum of all bitrates received). It penalises switching between

bitrates and initial and rebuffering delays are heavily penalised with a one second of buffering considered equivalent to a 3000kbps reduction in bitrate. This is useful as big jumps in quality is jarring for the end users and buffering extremely frustrating. Yin returns a value that is dependent upon the HAS rate chosen. The higher the value returned by Yin the better the QOE.

Finally this project uses the **P1203** library[20][21] This repository is an implementation of the ITU-Rec-P1203 standards of quality measurement for adaptive audiovisual content. This is a very useful measure as it not only incorporates a measurement of image quality but also auditory quality of the content. Bitrate and resolution are rewarded heavily in this model. P1203 returns values in the range of [0,5] with values closer to five yielding a better QOE.

The above QeMs provide a fantastic opportunity to analyse the quality of the user experience under different network conditions and compare results.

2.1.8 goDASHbed

goDASHbed is a testbed framework created for goDASH. GoDASHbed utilises Mininet in order to simulate multiple goDASH clients streaming over the same network. Each client generates their own debug and log files. The network can be modified as desired allowing you to add or remove nodes, change link types, modify delay on links, change link bandwidth and the number of clients. GoDASHbed also uses an existing 4G trace [22] to simulate real world network throughput variance making goDASHbed a more realistic simulation of real world conditions.

2.1.9 Mininam

Mininam is a Python visualisation tool that creates real time animation of traffic on a Mininet network using pythons TKinter library. Mininam provides an easy to use framework in which you simply pass the already created Mininet network to Mininam. Mininam then produces real time visualisation of the network traffic. Mininam provided an easy way to demonstrate the Cooperative streaming model to others and a way to ensure the network was working correctly without the need for using debug logs.

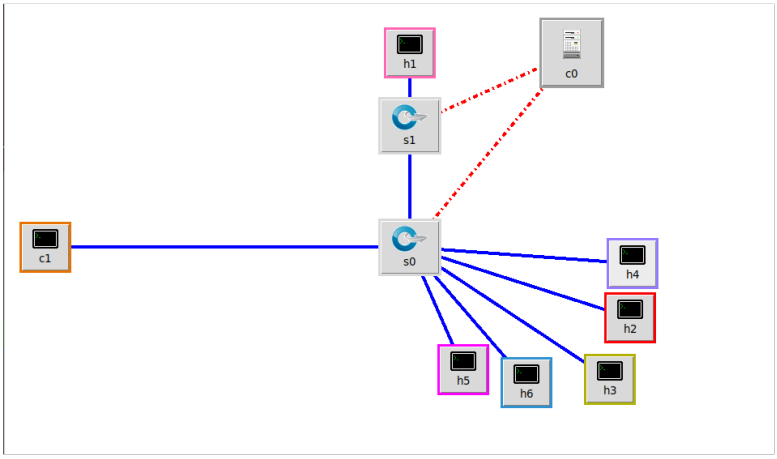


Figure 12: Network Topology in Mininam : c1:Consul Server, h1:Web Server, h2 - h6:Peers, s0-s1:Switches on the network

Figure 12 is an example of the Cooperative Peer-to-Peer streaming network with 5 clients running in Mininam. This gives a clear picture of the network structure with consul and the video streaming clients all connected on one switch while the web server is isolated over a link designed to replicate requesting video content of the internet.

2.2 Summary

Chapter 2 has discussed and explained the main technologies used in the implementation of this project. Understanding of these technologies will allow for better comprehension of the design and implementation decisions made during this project. Chapter 3 will show how these technologies were utilised to create the Cooperative Peer-to-Peer Video Streaming network.

3 Design and Implementation

This chapter will discuss the design of the Cooperative Peer-to-Peer Video Streaming network and walk through the implementation process explaining the steps taken to create this Cooperative network along with any issues that occurred during its creation.

3.1 Design

As discussed in chapter 2 it was decided that the Cooperative Peer-to-Peer Video Streaming network would take the shape of a Hybrid Peer-to-Peer network with Hashicorp's Consul software acting as the local server, GRPC providing communication between clients and goDASH acting as the clients DASH player. This section will now discuss and explain the design of the Cooperative Peer-to-Peer Video Streaming network.

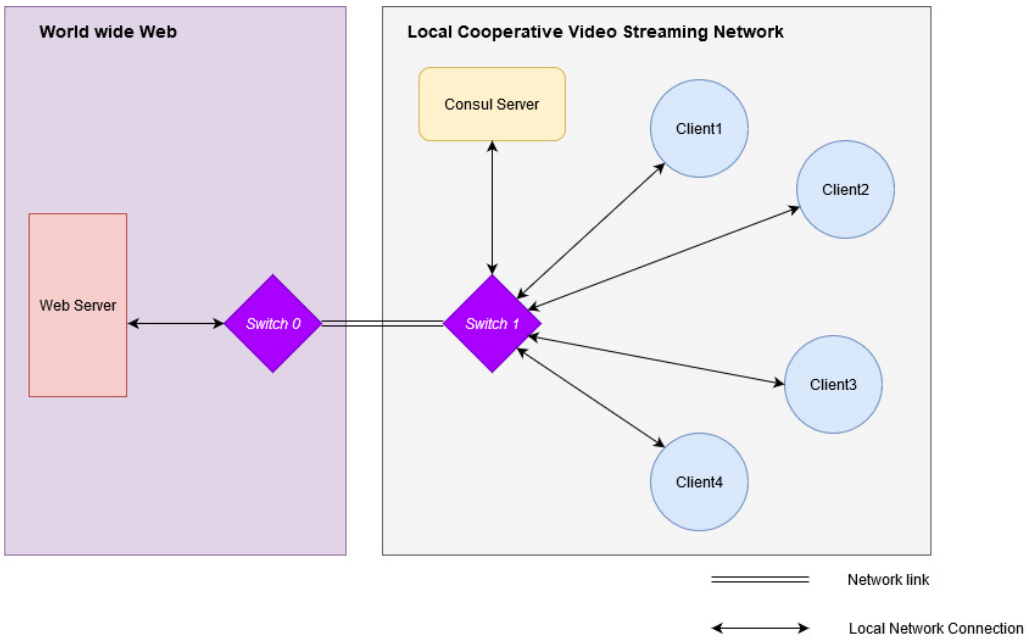


Figure 13: Diagram of showing the network structure of the Cooperative Peer-to-Peer Video Streaming model

3.1.1 Local Server (Consul)

Consul will act as the local server in this network tracking what Peers and video content is available on the network. It tracks the Peers and video content using Consul's Key-Value system. The Key for this Key-Value pairing consists of the Original URL of the video content with the port address of the hosting peer appended to the end of the URL to ensure all Key entries are unique even in the case where multiple Peers are hosting the same video content. The Key is stored as a string. The Value consists of the GRPC server address of the device hosting the corresponding video content. The Value entry is stored as bytes. The Key-Value pairs can be added or retrieved using the *GET* and *PUT* methods provided by the Consul KV API. The port address appended to the end of the Key will not affect retrieval as Consul's *GET* method searched by prefix ensuring all devices hosting a particular piece of video content will be retrieved. These *GET* and *PUT* allow Peers to search for content on the network and update Consul when new content is available on the network.

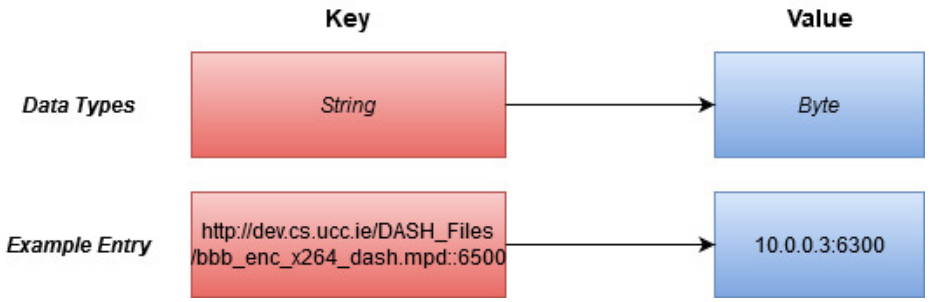


Figure 14: Diagram of showing the structure of Consul Key-Value pairs and an example of a KV entry

3.1.2 Peers

Components: Every Peer in the network consists of four major components. These components and their purposes are as follows:

GRPC Server: The GRPC server allows other Peers in the network to make requests when searching for content between Peers.

GRPC Client: The GRPC client can make GRPC requests to other clients when searching for video content on the network.

Web Server: Each Peers web server hosts all content that each Peer has already downloaded. This allows this previously downloaded content to be readily available for other Peers to download.

goDASH: Each Peer uses the goDASH player. GoDASH allows Peers to stream DASH video content over the network.

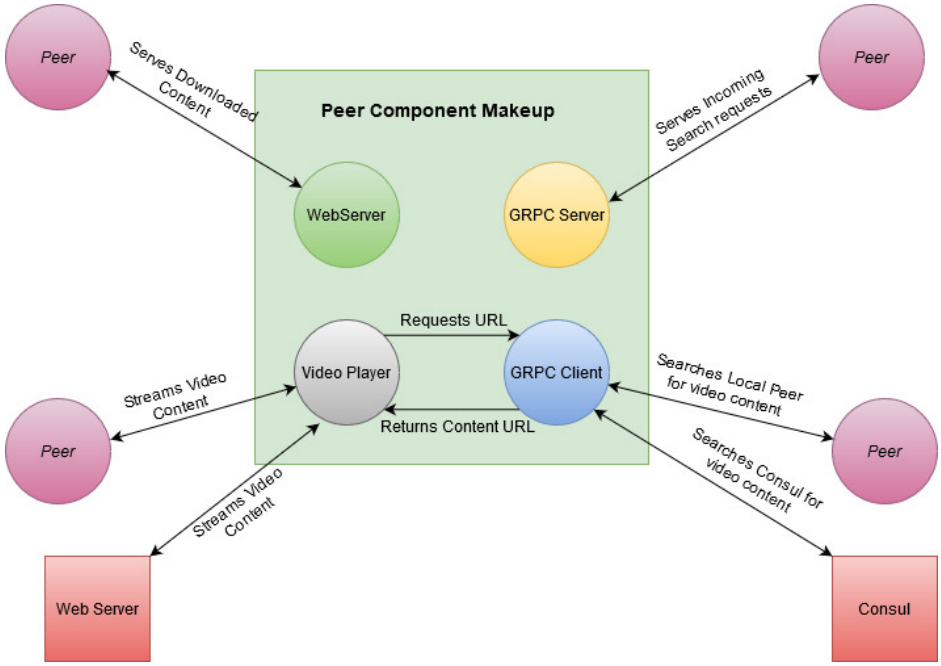


Figure 15: Diagram of a Cooperative Video Streaming Peer showing the Peers individual components and their corresponding functions

Data Structures: Each Peer maintains and tracks a few key pieces of data. This data is used to request and serve content to and from other Peers.

Known Clients: Known Clients is a list that tracks the GRPC Server address of Peers that have been previously contacted. When searching for content on the network a Peer first searches all Peers in the Known Clients list.

Video Content Currently Downloaded: This is a map of all the content a Peer has previously downloaded. This list is used when receiving requests for content from other Peers. When a search request is

received this list is consulted to check if the content is present on this Peer. The use of the map data structure allows for fast and efficient lookup.

Consul Address: This is the preset IP address of the local Consul server.

Hosting Content: In order to allow for compatibility with goDASH player the simplest solution was to have each Peer host previously downloaded video content. This allows for the use of goDASH with minimal modifications to the goDASH code base. GoDASH requests video content using a HTTP GET request. The Cooperative Peer-to-Peer Video Streaming network now must simply return the URL of the video content hosted on one of the networks Peers allowing goDASH to function as normal. This saves time during the implementation of the project while also allowing for a fairer comparison between the modified Cooperative version of goDASH and goDASH's standard implementation.

Searching the Network: When searching the network for video content Peers first consult all Peers they have previously contacted. This is done using a predefined GRPC request. When a Peer receives a search request the Peer first consults the map of all previously downloaded content. If the requested video content is present, the Peer will return the URL of the video server and file hosting the video content. If the requested content is not present the Peer then sends another predefined GRPC search request to all of its known Peers. If the content is present among its neighbors the neighbor hosting the requested video content will return the URL of the web server that hosts the desired video content.

If the content can not be found among known peers in the network the Peer will then request the video content from Consul. If the video content is present anywhere in the network Consul will return the GRPC server address of the Peer that hosts the video content. This Peer will then receive a request from the searching Peer for the desired video content. This Peer then returns the URL of the video content to the searching Peer ready for streaming. If the video content is not present anywhere in the network the Peer will stream the video content from the internet as normal.

After the video content is streamed the Peer will inform Consul using the *PUT* method that new content is available from this Peer.

Algorithm 1 Search(URL) PseudoCode

```
URL ← String URL of requested video content
KnownPeers ←List of addresses of Known Peers
Consul ← Address of local Consul Server
if length(KnownPeers) > 0 then
  for peerinKnownPeers do
    contentURL ← peer.Search(URL)
    if contentURL ≠ null then
      RETURN contentURL
    end if
  end for
end if
contentURLList ← Consul.GET(URL)
if length(contentURLList) > 0 then
  contentURLLISt = random.Shuffle(contentURLList)
  peerGRPCAddress = contentURLList[0]
  contentURL = searchRequest(peerGRPCAddress, URL)
  if contentURL ≠ null then
    KnownPeers.append(peerGRPCAddress)
    RETURN contentURL
  else {contentURL is empty}
    RETURN URL
  end if
else {Consul does not know the content location}
  RETURN URL
end if
```

3.1.3 Deliverables and Requirements

After the general design of the Cooperative Peer-to-Peer Streaming network and its function was decided on a set of deliverables and their corresponding requirements was created they are as follow:

1. Create Cooperative clients in Golang that are capable of searching the Peer-to-Peer network and retrieving desired content from other Peers.
    - Connect to Consul Server on localhost
    - Perform Get and Put requests to the Consul server

- Each Peer must be able to make a receive GRPC requests for desired information(such as IP address)
- Each Peer must host a Content server capable of distributing video segments when requested
- Each Peer must maintain a list of currently known Peers
- Each Peer must be able to search its list of known Peers for desired video content

2. Integrate the Cooperative Clients into GoDASH to allow for use of DASH player and its algorithms as well as the use of QOE metrics.
    - GoDASH must be able to stream content while using the Cooperative Client network
    - GoDASH QOE metrics must be correct despite modification of using the Cooperative clients
    - GoDASH must be able to receive content from each Peers content server when requesting content
3. Integrate the Cooperative Clients version of GoDASH with GoDASHbed.
    - Cooperative Clients can communicate with consul over simulated goDASHbed network
    - Cooperative Clients can communicate with each other over the simulated goDASHbed network
    - goDASHbed Clients can send and receive correct video content over the network using DASH algorithms
    - goDASHbed network set up with realistic network delays and bandwidth links
4. Add Mininam visualisation to goDASHbed
    - Correct real time visualisation of traffic on the Cooperative network is seen

3.2 Implementation

This section will now discuss the implementation and completion of each Deliverable in this project beginning with the initial development environment configuration and continuing to the final implementation of the Cooperative Peer-to-Peer Video Streaming network.

3.2.1 Creation of Cooperative clients in Golang that are capable of searching the Peer-to-Peer network and retrieving desired content from other Peers

The first task in the implementation of this project was to create the development environment. GoDASH first needed to be installed on the local machine. GoDASH includes an easy to follow description of how to correctly install the goDASH software. GoDASH was installed successfully on the local machine and streamed video content as expected. However an issue occurred when attempting to use the QOE metrics included in goDASH causing goDASH to crash. After much debugging it was found that the P.1203 library used for the QOE metrics was the issue. The error was caused by an incompatibility between Windows 10 and the python multiprocessor library that the P.1203 library relies on. To resolve this issue a dual boot system was created with the Ubuntu linux distribution on the new partition. Once using Ubuntu goDASH functioned as expected. To run goDASH the following command is used:

```
./goDASH -config ../config/configure.json
```

Listing 1: Running goDASH from command line

The installation of Consul was the next step and was seamless, requiring the download and running of an installer that completed the installation automatically. To test Consul was running as needed the below command was used:

```
consul agent -dev
```

Listing 2: Running Consul from command line in dev mode

This command runs Consul on localhost. The -dev argument prevents Consul from saving its server state between runs making it ideal for testing and debugging.

To begin to implement this Cooperative network the protocol buffers must first be defined. Defining these protocol buffers will create an interface for the

functions that each GRPC server and client must implement. These functions will allow GRPC clients and servers to communicate over the network. These functions were defined in the file P2PService.proto.

```
service P2PService {
    rpc CheckClients (CheckRequest) returns (CheckReply){}
    rpc GetServerAddr (ServerRequest) returns (
   ServerRequestReply){}
    rpc SecondCheckClient (SecondCheckRequest) returns(
   SecondCheckReply){}
}
```

Listing 3: Protocol buffer rpc definitions

The functions made available and their explanation were as follow:

- **CheckClient:** This function takes a CheckRequest message as input. The CheckRequest message contains the address of the URL that is being requested. The CheckRequest function checks the nodes previous downloads for the video content being requested. If it is not present locally it loops through all its Known Peers sending a SecondCheckClient request. If the video content is present locally or in one of the Known Peers CheckClient returns a CheckReply to the requesting peer with the GRPC server address of the device that has the video content. If the content is not found the fucntion returns a CheckReply message with a null address.

- **SecondCheckClient:** This function takes a SecondCheckRequest message as input that contains the URL of the video content that is being requested. If the content is present on the peer SecondCheckClient will return a SecondCheckReply message with the address of the peers GRPC server attached. Otherwise the function will return a SecondCheckReply with an empty address.

- **GetServerAddr:** This function takes a ServerRequest message as input. The function returns a ServerRequestReply with the address of the peers web server that hosts the previously downloaded video content.

Once the protocol buffers were defined it is then necessary to generate the data access classes using the below command.

```
protoc -I P2PService/ P2PService/P2PService.proto --go_out=
  plugins=grpc:P2PService
```

Listing 4: protoc command to generate data access classes

These classes provide a server and client interface that each peers GRPC server and client will implement. The implementation of these interfaces allow the peers to communicate using GRPC.

The next step in the implementation of the Cooperative Peer-to-Peer Video Streaming network is the creation of the Peer class itself. This class called NodeUrl.go acts as the peers in the Cooperative network. The struct of the NodeUrl class looks as follows:

```
type NodeUrl struct {
 //node variables
 ClientName      string
 url             string
 previousUrl     map[string]string
 Addr            string
 ContentPort     string
 ContentLocation string
 Clients         map[string]string
 IP net.IP
 Registered bool

 //consul variables
 SDAddress string
 SDKV      api.KV

 //Server for implementation
 pb.UnimplementedP2PServiceServer
}

```

Listing 5: NodeUrl struct

The key data stored in the NodeUrl struct is, as discussed in the design section, the KnownPeers list (called clients in the class) and the map of previously downloaded content (called PreviousUrl in the class). The Key functions of the NodeUrl class are as follows:

- **StartListening:** This function starts a GRPC server on the assigned network port. The code for this function is provided in the Appendix Section A. This function first creates a listener on the assigned network port. The function then creates an instance of a GRPC server. This new GRPC server is registered with our protocol buffer implementation by passing the GRPC server to the generated RegisterP2PServiceServer function. Finally the listener created at the start of the function is passed to the GRPC server. The GRPC server will now serve all of the predefined GRPC functions. In the case where the server fails to serve a client the server is stopped.

- **ContentServerStart:** Starts a HTTP web server serving the content that the peer has previously downloaded.The code for this function is provided in the Appendix Section B. This function takes the content location and network port as input. First the function creates a instance of a Golang http server using the Golang http library. The server is instantiated with the default configuration. Next the function creates a handler that serves the files present in the provided file location. This new handler is then passed to the server instance allowing the server to serve files from the given file location. Finally the the server and network port are passed to the listen and server function which starts the server on the given network port.

- **Search:** The Search function takes the requested URL as input. The code for this function is provided in the Appendix Section C. First it processes the given URL. It takes the file base of the URL as the files on the Peers are stored with just the file base. Next we hash the given URL to get the key used to search the network. The function then checks that the content is not already present on the local peer. If the content is not present the function moves to searching its known neighbors.

  The function loops over all known peers. For each peer it first dials and establishes a GRPC connection. Using the established GRPC connection the peer invokes the GRPC CheckClients function providing the target key as input. If the response is not empty the GRPC GetContentAddress function is utilised over the same GRPC connection. GetContentAddress returns the IP address of the device that hosts the desired content. The URL hosting the content is then constructed us-

ing the hosting peer's IP address and the filebase processed earlier. The content can be downloaded from the returned URL. If the content is still not found the peer then searches the Consul server.

The peer first searches Consul using the previously established Consul connection. Consul is searched using the List function. The function searches by prefix. The list function returns a list of Key-Value pairs that match the given prefix (key). The KV pairs are first shuffled to ensure even distribution of download requests. The function then loops through all of the returned KV pairs. The peer then requests the content server address of the peer corresponding to the KV pair. If the content address request is successful the URL hosting the content is constructed and returned. If the content address request fails the function continues to loop through the KV pairs. In the case where all of the above searches fail to provide the desired content the function returns the original URL allowing the content to be downloaded from the internet as normal

To test the network a DASH Datasets for AVC (H.264) [23] made available on a University College Cork server was used. To begin testing the Consul server must first be instantiated. Then four Peers were created in four separate Linux terminals. The peers requested the same piece of video content. It was verified that the Peers were downloading the video content (single 4 second video segment) initially from the university server and then sharing the video content among themselves locally.

```
DEBUG: 2020/04/11 11:33:58 get file from URL: http://192.168.1.3:61644/bbb_1920x1080_60fps_4300kbps_segment4.m4s

DEBUG: 2020/04/11 11:33:58 setup default client but with a defined ssl security check
DEBUG: 2020/04/11 11:33:58 URL is : http://192.168.1.3:61644/bbb_1920x1080_60fps_4300kbps_segment4.m4s
DEBUG: 2020/04/11 11:33:58 Protocol is : HTTP/1.1
DEBUG: 2020/04/11 11:33:58 Before consul update
DEBUG: 2020/04/11 11:33:58 After consul update

DEBUG: 2020/04/11 11:33:58 grep being used on Linux

DEBUG: 2020/04/11 11:33:58 P1203 has the correct hex value

DEBUG: 2020/04/11 11:33:58 P1203 bitrate is 3895.527344

DEBUG: 2020/04/11 11:33:58 checking for P1203 compatibility

DEBUG: 2020/04/11 11:34:01 conventional has choosen rep_Rate 0 @ a rate of 39537
```

Figure 16: Output showing the returned URL of a Peer sharing content on local network

At this stage of the implementation of this project the requirements for the first deliverable had been achieved. The network could perform all of the following:

- Connect to Consul Server on localhost
- Perform Get and Put requests to the Consul server
- Each Peer must be able to make a receive GRPC requests for desired information(such as IP address)
- Each Peer must host a Content server capable of distributing video segments when requested
- Each Peer must maintain a list of currently known Peers
- Each Peer must be able to search its list of known Peers for desired video content

The project then progressed to the next stage which was integration with the goDASH software.

3.2.2 Integration the Cooperative Clients into GoDASH to allow for use of DASH player as well as the use of QOE metrics.

First step was to instantiate a NodeUrl instance in goDASH with the correct arguments. To facilitate this command line arguments were added to goDASH. One argument contains the location of the download folder containing the previously streamed content for redistribution, the other is the peer name this is need for testing purposes only so the peers being run during testing can be differentiated. The NodeUrl class is instantiated in the main function of the goDASH as follows:

```
1     //server address passed to variable s from command line
      arguments
2   s := strings.Split(os.Args[4], "/")
3
4   //initialise variables of NodeUrl
5   Noden := P2Pconsul.NodeUrl{
6     ClientName:      s[len(s)-1],
7     ContentLocation: os.Args[4],
8     Clients:         nil,
9     SDAddress:       "localhost:8500",
```

```
10
11      //randomly assign content port to avoid clashing with
        other clients on localhost
12      ContentPort:     ":" + strconv.Itoa(rand.Intn(63000)
        +1023),
13  }
14
15  //initialise Noden components
16  Noden.Initialisation()
17
18  //pass reference to Node to http library
19  http.SetNoden(Noden)
```

Listing 6: Instantiation of NodeUrl class in goDASH

As seen in the snippet of code above the instance of NodeUrl is then passed the the http package. This reference is used after the content has been successfully downloaded using a HTTP GET request to update Consul on the new content that is now available from this peer.Originally the NodeUrl reference was also used to search the network in the http package. However adding the search to the http package caused issues in the measuring of the QOE metrics. To resolve this the NodeURL search was moved to the stream function. In the main function another reference to the NodeUrl is passed to the main stream function. In the stream function NodeUrl.Search is called. The NodeUrl instance will then search the local peer-to-peer network for the requested video content. If the content is present on the network the modified URL pointing to the peer in the network with the requested content is returned. If the content is not present on the network the original URL is returned and the video content is downloaded from the internet as normal.

```
1     // get the current url - trim any white space
2   currentURL := strings.TrimSpace(urlInput[mpdListIndex])
3   logging.DebugPrint(debugFile, debugLog, "\nDEBUG: ", "
      current URL header: "+currentURL)
4
5   logging.DebugPrint(debugFile, debugLog, "DEBUG: ", "Made to
      consul search")
6
7   //search the local network for video content with Original
      URL of currentURL
8   currentURL = Noden.Search(currentURL)
9   logging.DebugPrint(debugFile, debugLog, "DEBUG: ", "Consul
      return URL: "+currentURL+"\n")
10
```

```
11  //remove search time from url
12  l := strings.Split(currentURL, "::")
13
14  //assign currentURL to url of video content
15  currentURL= l[0]
16  logging.DebugPrint(debugFile, debugLog, "DEBUG: ", "Parsed
     url: "+currentURL+"\n")
```

Listing 7: How NodeUrl Search is called in goDASH stream function

The modified Cooperative version of goDASH was tested similarly to the last deliverable. Consul was run in one terminal while the evaluate method of goDASH was used to create multiple goDASH instances downloading the same content simultaneously.

Seg_#	Arr_time	Del_Time	Stall_Dur	Rep_Level	Del_Rate	Act_Rate	Byte_Size
1	140	131	0	237	2998	98	49093
2	3724	1244	0	2358	16612	5166	2583281
3	5565	864	0	4282	28614	6180	3090357
4	8304	30	0	4282	532434	3993	1996631
5	10939	7	0	4282	2265221	3964	1982069
6	14009	5	0	4282	1496104	1870	935065
7	17624	642	0	4282	19884	3191	1595755
8	22137	1425	0	4282	17336	6176	3088059
9	25372	5	0	4282	4031468	5039	2519668
10	28713	8	0	4282	2260755	4521	2260755

Figure 17: Single client output of cooperative network

P.1203	Clae	Duanmu	Yin	Yu
1.878	0.000	46.465	-11762.149	0.238
3.286	0.000	46.173	-23524.298	1.298
3.632	0.031	53.067	2834.590	2.293
3.828	0.839	56.548	7116.925	2.790
3.969	1.438	58.637	11399.260	3.089
4.084	1.886	60.029	15681.595	3.288
4.283	2.233	61.024	19963.930	3.430
4.366	2.509	61.770	24246.265	3.536
4.417	2.736	62.350	28528.600	3.619
4.459	2.925	62.814	32810.935	3.686

Figure 18: QOE ouput of single client in the cooperative network

During testing it was verified that this deliverable fufilled the following requirements.

- GoDASH must be able to stream content while using the Cooperative Client network
- GoDASH QOE metrics must be correct despite modification of using the Cooperative clients
- GoDASH must be able to receive content from each Peers content server when requesting content

The next stage of development was to move the modified Cooperative goDASH to a virtual environment to allow for realistic testing to be performed. GoDASHbed offers a virtual testing framework for goDASH and will be modified to work with the Cooperative version of goDASH.

3.2.3 Integrate the Cooperative Clients version of GoDASH with GoDASHbed

The first stage of this deliverable was to successfully install goDASHbed. GoDASHbed has very good step by step installation instruction making it easy to set up and install. The video content (45GB) was then downloaded locally to allow for the simulation of video streaming in the virtual network. Once the standard goDASHbed implementation was up and running it was then time to begin integrating the modified version of goDASH into goDASHbed. Firstly a Consul server was added to the network along with the necessary network links. Consul and all the peers in the network were connected to the same network switch. To run Consul on the simulated network the following command was used.

```
1     print("starting consul")
2     #start consul in development mode on IP address
    10.0.0.2:8500
3     ttt = consul.cmd("consul agent -dev -client 10.0.0.2 &")
4     sleep(5)
```

Listing 8: Consul being called on a device in the simulated network

This command runs a Consul server on the predefined IP address. This IP address is preset in the Cooperative goDASH implementation. Next was to

modify the path from the standard implementation of goDASH to the Co-operative Version of goDASH. This also required the addition of the correct command line inputs for both the goDASH instances name and the corresponding location of the previously downloaded.

```
#start godash client with parameters, config file location
    and the location to be served by the content server
cmd = params["cwd"]+"/../goDash/DashApp/src/goDASH/goDASH -
    config " + \ params["output_folder"]+"/R"+str(run)+params[
    "current_folder"]+
config_folder_name+client_config+\
" -serveradd"+params["output_folder"]+"/R"+str(run)+params[
"current_folder"]+log_folder_name+"/"+dash_client_name
```

Listing 9: goDASH being called from goDASHbed

Once these modification were complete the modified version of goDASH was ready for testing using goDASHbed. During testing an issue was discovered where goDASH would not complete the download of content when run with more than six simultaneous clients. The issue was caused by Consul being unresponsive to any form of request. To check Consuls functionality it was tested again outside of the virtual network on localhost. When tested on localhost with ten simultaneous clients it functioned as expected ruling out an issue with the Consul server itself. Therefore the problem was with the virtual network. After much debugging it was found that the Consul server had insufficient bandwidth to serve all the client requests. This was puzzling as Consul was configured with a high bandwidth connection on a low latency link. After further digging it was discovered the throttleLink function that limits the connection bandwidth for video clients based on a 4G trace was accidentally throttling the Consul server connection also. To remedy the situation an extra switch was added to the network. This switch acts as a relay between the Consul server and the local network switch and has no delay on the link. The addition of this extra switch removes Consul from the scope of the throttleLink function while not impairing Consuls performance allowing a local connection to still be simulated correctly. Furthermore during this stage of testing it was discovered that the secondCheck function was very inefficient. When a peer receives a request to search for video content it first checks if it is currently hosting the requested content. If the peer does not have the requested content the peer would then use the secondCheck function to loop through the peers neighbors checking for the requested content. While this second layer of searching made finding the requested content in

the network more likely it caused a large amount of video stalling due to the added complexity. For instance in a network of 10 peers with 10 neighbors each, using the secondCheck method would cause 100 search requests for each search request. Removing the secondCheck function vastly increased the Cooperative models efficiency, reducing some request times from 9 seconds to just 700 milliseconds. The model will now only search its known neighbors. GoDASH now functioned correctly in the simulated network producing the correct logs and output. The project is now almost ready for testing.

```
DEBUG: 2020/04/11 13:58:48 Testbed in use
DEBUG: 2020/04/11 13:58:48 loading X509 key and cert: /home/jpos2/go/src/goDash/DashApp/src/goDASH/http/certs/cert.pem /home/jpos2/go/src/goDash
DEBUG: 2020/04/11 13:58:48 reading X509 cert
DEBUG: 2020/04/11 13:58:48 creating tls config
DEBUG: 2020/04/11 13:58:48 creating our http transpppppport using our tls config
DEBUG: 2020/04/11 13:58:48 creating our client using our http transport and our tls config
DEBUG: 2020/04/11 13:58:49 URL is : http://10.0.0.3:24122/tearsofsteel_enc_x264_dash_init.mp4
DEBUG: 2020/04/11 13:58:49 Protocol is : HTTP/1.1
DEBUG: 2020/04/11 13:58:49 Before consul update
DEBUG: 2020/04/11 13:58:49 After consul update

DEBUG: 2020/04/11 13:58:49 We are using repRate: 12
DEBUG: 2020/04/11 13:58:49 We are using : conventional for streaming
DEBUG: 2020/04/11 13:58:49 current segment baseURL+segURL: 320x180/235kbps/tearsofsteel_320x180_24fps_235kbps_segment1.m4s

DEBUG: 2020/04/11 13:58:49 in consul search url :http://www.goDASHbed.org/content/tos_4sec_main/x264/tearsofsteel/DASH_Files/main/320x180/235kbp

DEBUG: 2020/04/11 13:58:49 Start of consul search Location:tearsofsteel_320x180_24fps_235kbps_segment1.m4s

DEBUG: 2020/04/11 13:58:49 Looping client check

DEBUG: 2020/04/11 13:58:50 current URL joined: http://10.0.0.3:24122/tearsofsteel_320x180_24fps_235kbps_segment1.m4s::clients::668.310014ms

DEBUG: 2020/04/11 13:58:50 current URL joined: http://10.0.0.3:24122/tearsofsteel_320x180_24fps_235kbps_segment1.m4s

DEBUG: 2020/04/11 13:58:50 current URL joined: tearsofsteel_320x180_24fps_235kbps_segment1.m4s
DEBUG: 2020/04/11 13:58:50 get file from filebase prior: tearsofsteel_320x180_24fps_235kbps_segment1.m4s

DEBUG: 2020/04/11 13:58:50 get file from currentURL: http://10.0.0.3:24122/tearsofsteel_320x180_24fps_235kbps_segment1.m4s

DEBUG: 2020/04/11 13:58:50 get file from filebase: tearsofsteel_320x180_24fps_235kbps_segment1.m4s

DEBUG: 2020/04/11 13:58:50 get file from URL: http://10.0.0.3:24122/tearsofsteel_320x180_24fps_235kbps_segment1.m4s

DEBUG: 2020/04/11 13:58:50 Testbed in use
DEBUG: 2020/04/11 13:58:50 loading X509 key and cert: /home/jpos2/go/src/goDash/DashApp/src/goDASH/http/certs/cert.pem /home/jpos2/go/src/goDash
DEBUG: 2020/04/11 13:58:50 reading X509 cert
DEBUG: 2020/04/11 13:58:50 creating tls config
DEBUG: 2020/04/11 13:58:50 creating our http transpppppport using our tls config
DEBUG: 2020/04/11 13:58:50 creating our client using our http transport and our tls config
DEBUG: 2020/04/11 13:58:50 URL is : http://10.0.0.3:24122/tearsofsteel_320x180_24fps_235kbps_segment1.m4s
DEBUG: 2020/04/11 13:58:50 Protocol is : HTTP/1.1
DEBUG: 2020/04/11 13:58:50 Before consul update
DEBUG: 2020/04/11 13:58:50 After consul update

DEBUG: 2020/04/11 13:58:50 grep being used on Linux

DEBUG: 2020/04/11 13:58:50 P1203 has the correct hex value

DEBUG: 2020/04/11 13:58:50 P1203 bitrate is 8.882812

DEBUG: 2020/04/11 13:58:50 checking for P1203 compatibility
```

Figure 19: Log output of a video client in goDASHbed network

It has fulfilled all the requirements of this deliverable. These requirements are as follows:

- Cooperative Clients can communicate with consul over simulated goDASHbed network

- Cooperative Clients can communicate with each other over the simulated goDASHbed network
- goDASHbed Clients can send and receive correct video content over the network using DASH algorithms
- goDASHbed network set up with realistic network delays and bandwidth links

The final step is to add a visualisation of the virtual network in order to make it easier to debug during testing and to provide a visual reference when demonstrating this project to others at a later date.

3.2.4 Addition of Mininam visualisation to goDASHbed

The implementation of virtual network visualisation was seamless with MiniNam. To install you must simply clone the MiniNam master branch and move the file MiniNam.py to your project directory. Then to use Mininam you simply place the following command just before your Mininet network has terminated.

```
Mn = MiniNAM(tk.Tk(),600, 1000 , net, locations={"c0
    ":(700,60),"c1":(50,300),"h1":(500,60),"s1":(500,150),"s0
    ":(500,300),"h2":(300,300)})
```

This simple piece of code gives a live representation of the traffic on your virtual network. This was a great aid in ensuring the traffic on the network was moving as expected.

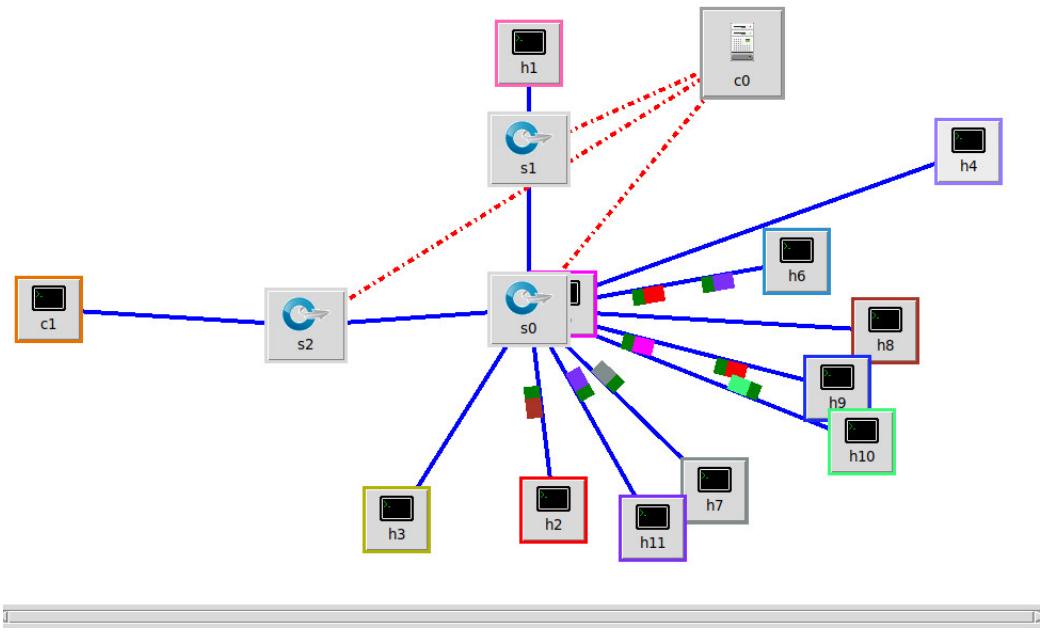


Figure 20: Output of the MiniNam network visualisation

This chapter walked through how this project was designed and implemented. With a working implementation of the Cooperative Peer-to-Peer Video Streaming network complete it is now time to test the effectiveness of this network. Chapter 4 will walk through the testing and evaluation of this network.

4 Evaluation

This chapter will give an evaluation of the project process, determining whether the original project goals were achieved by the final implementation of the project. This section will also walk through what steps taken aided project development as well as those that hindered the project and should be improved in future projects.

4.1 Project Goals

The goal of this project was to create a Cooperative Peer-to-Peer Video Streaming network using goDASH. The network was to be capable of streaming DASH content from the internet as well as being capable to stream previously downloaded video content to/from other peers while maintaining or improving QOE. To evaluate whether these goals were met the final implementation of the project will be explained in full to ensure each aspect of the goal is achieved.

In the final implementation of the Cooperative Peer-to-Peer Video Streaming network consisted of two main components, the local server and the peers. Hashicorps Consul Key-Value system runs on the local server. The server maintains a record of all content that is hosted on each device in the network. Each Key is the URL of the video content being requested and each corresponding Value is the address of the GRPC server of the peer hosting the video content. The local server only supports two functions. The GET function allows peers to search Consuls records for a URL corresponding to the desired video content by prefix. This will return a list of every device hosting the requested content. The PUT function allows peers to update Consuls records with new Key-Value pairs when new content is available from that peer.

Each peer in the network consists of four components, the web server, the GRPC server, the GRPC client and the goDASH player. The web server makes all previously downloaded content available for other peers to download over HTTP. The GRPC server responds to GRPC search requests from other peers. When an incoming GRPC search request is received by a peer the peer searches all previously downloaded video content for the requested content. If the requested content is present the peer will return the URL for the requested video content hosted on the peers web server.

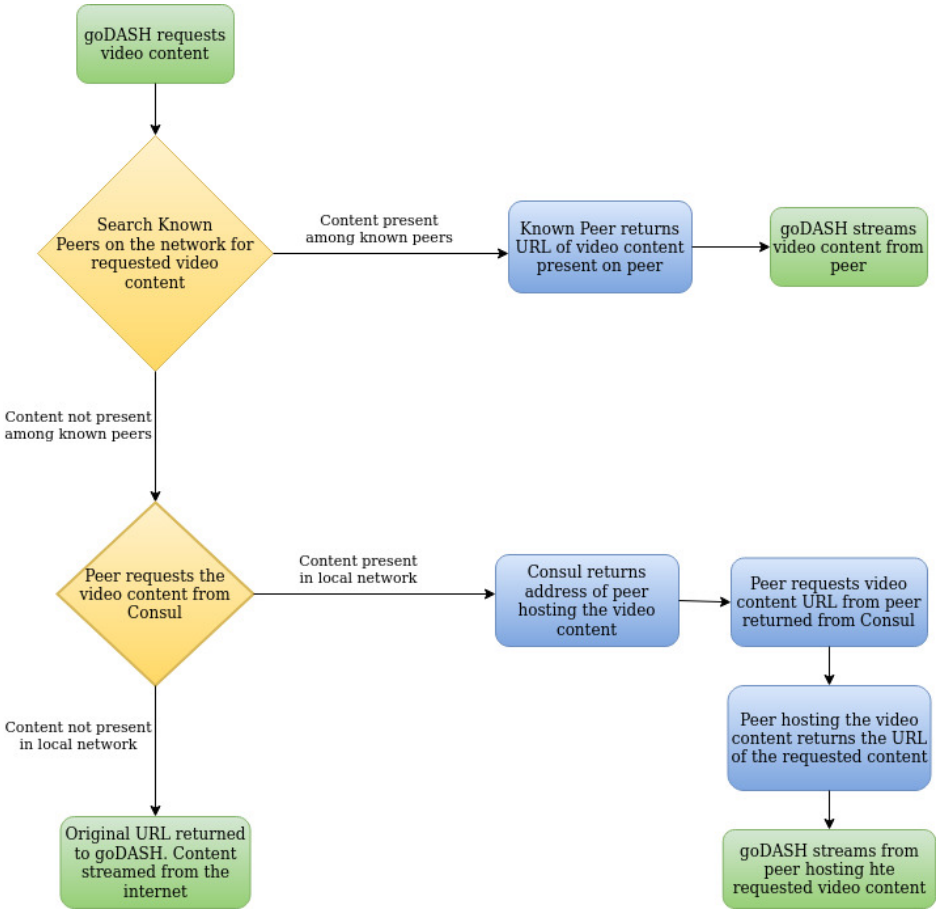


Figure 21: Flowchart of the process for streaming a video segment

GoDASH is a DASH player. GoDASH is responsible for requesting the correct video quality for the next segment of video based on currently available resources. When goDASH requests a piece of video content it is first passed to the GRPC client. The GRPC client will search the local network for the requested content before goDASH requests the video content from the internet. The GRPC client will send a search request to every known peer in the network. If any known peer has the requested content the known peer will return the URL to the web server of the peer hosting the requested content to the GRPC client. The GRPC client will then return this URL to goDASH to stream the video content. If the video content is not present among the known peers the GRPC client will request the content from the

Consul local server using a GET request. If the content is present Consul returns the GRPC server address of the peer hosting the video content. The GRPC client then sends a request to the peer hosting the video content for the URL of the requested content. The peer will return the corresponding URL to the GRPC client. The GRPC client will then return the URL to goDASH for streaming. If the video content is not found during either search the GRPC client will return the original URL of the requested content to goDASH and goDASH will stream the video content as normal from the internet. The above process is illustrated in figure 21.

The search process outlined above clearly shows that the network has met the goals set at the outset of this project. It is capable of streaming video content from the internet, streaming previously downloaded video content to/from other peers and as will be shown in chapter 5 maintains or improves QOE. Since the goal of this project has been achieved this project can be deemed successful. While the project was successful in its goal there is still room for improvement with regards project development process. In the next section we will explore some positive aspects of the project process as well as those aspects that should be worked upon in upcoming projects.

4.2 Process evaluation

This section will list and explain both the good aspects of the project process as well as the aspects that hindered the progress of the project.

4.2.1 Positive aspects

- **Model Selection** The model selection for this network was highly effective allowing for the network to be easily implemented in existing infrastructure. Furthermore the configuration of the model can be scaled quite easily allowing for the local server to serve a single home, a shopping center or perhaps a whole town. The model was also attainable given the time constraints of the project. While a true Peer-to-Peer network would be better suited to immediate adoption in current networks its complexity made it unattainable. Utilising the Hybrid Peer-to-Peer model made the project attainable while fulfilling the needs of the project completely. Finally the model was highly effective giving video client a QOE on par or even exceeding that of the standard DASH configuration (Discussed in detail in chapter 5).

- **Tool Selection** Another positive aspect of this project was the tool selection for this project. GoDASH was a great project to work with offering not only an easy to use DASH player but also a fantastic QOE evaluation system perfect fro evaluating the effectiveness of this project. Consul was a fantastic product to work with offering an efficient and effective API. Consul required minimal configuration and ran smoothly throughout the project. GRPC was also a highly effective tool. While more complex to set up and use then the previous two tools mentioned once set up correctly was highly effective an provided highly efficient communication between peers in the network.

4.2.2 Negative aspects

- **Plan Code** The creation of this project involved the coding of many functions that were often highly complicated. Often during the course of the project some functions and coding decisions were created on the go without much consideration. This sometimes led to code not working or requiring the rewriting of the function altogether. This wasted precious time during the course of this project. Better planning of each function prior to starting to write code would have avoided this loss of time.

- **Comment Code** Another aspect of the project process that could be improved is the use of comments. During the course of this project many functions were created. While most had comments often times they were uninformative or the way in which the function operated had changed since. This led to time being lost divining the purpose of the function. This loss of time could be avoided by properly creating and maintaining the comments of the functions in the project.

By maintaining the positive aspects of the project going forward and improving the aspects that hindered the project the project process can be improved going forward. Along with improving the process the project itself can also be improved. The next section will focus on aspects of the project that could be improved going forward along with future work that could be completed on the network.

4.3 Project Improvements

This section will look at aspects of the project that can be improved along with future additions that would improve the functionality of the project.

- **Throttle function** Fix the incorrectly configured throttlelink function in goDASHbed. Currently it mistakenly throttles both the local server and the peers in the network. The current solution is to add an extra switch between the local server and the local network switch in the virtual network. This removes the local server from the scope of the throttlelink function but it is a temporary fix. The throttle function needs to be reconfigured to form a more permanent solution. This was outside the scope of this project due to time constraints.

- **Peers Leaving the Network** The current implementation assumes that no peers leave the network at any point. This is unrealistic. The creation of a way for peers to gracefully leave the network and update the local server appropriately would make the Cooperative Peer-to-Peer Video Streaming network far more functional.

- **Content Distribution** In the current implementation of the project it is assumed that all content that has been downloaded by a peer in the network will always be present on that peer. Future implementations should allow peers to delete content from peers without issue while also prioritising the storing of more popular video content. This would vastly improve the amount of storage used on peers in the network.

- **Live Content** This project focused on Video on Demand. Future projects could test this projects implementations effectiveness when dealing with distributing live content on the local network where minimising delay is far more important. The network could also be modified to be more efficient for live content expanding the applications of this cooperative streaming network.

This chapter evaluated the success of the overall project. This project was an overall success meeting all of the original goals of the project. While the project was a success this chapter also outlined improvements that can be made the both the project process and the project implementation itself in future work. The next chapter will analyse the performance of the Cooperative Peer-to-Peer streaming Network when compared to the standard DASH implementation.

5 Analysis

In this chapter we will analyse the effectiveness of the Cooperative Peer-to-Peer Video Streaming goDASH implementation in comparison to a standard goDASH implementation. Both configurations will be tested in a variety of network conditions that will be simulated using the modified goDASHbed implementation created during this project. The QOE metrics produced by these tests will then be analysed to form a conclusion as to the effectiveness of the Cooperative network in different networking environments. For the remainder of this chapter the original version of goDASH using the standard DASH implementation will be referred to as the Standard model or network while the modified Cooperative version of goDASH will be referred to as the Cooperative model or network.

5.1 Testing Network

This section will describe the structure and design of the virtual network used for testing. Full results of all test cases are attached in appendix D.

5.1.1 Structure

The structure of the network will be the same across all tests to enable a fair comparison. The network will contain one web server that hosts all available video content this web server is connected to switch1 in the network with a 10mb link. Switch1 is connected to switch0 over a network link, the bandwidth available on this network link will be varied depending on the amount of bandwidth needed for each test. This link between switch0 and switch1 is the bottleneck link in the network. All peers in the network are connected to switch0 over a 10mb link. In the Cooperative model the local server will also be connected to switch0 with a 10mb link. This network structure can be seen in figure 22. This is the network structure for all testing unless otherwise stated.

5.1.2 Latency

Latency is an important element of the network configuration to get right. The latency in a network link can largely determine the bandwidth capacity of the link. The average local area ethernet connection has a latency of

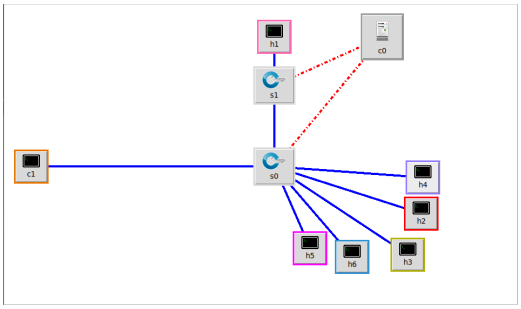


Figure 22: Virtual Network Structure: c1:Consul Server, h1:Web Server, h2 - h6:Peers, s0-s1:Switches on the network

about 10ms [24]. As a result all network links between peers and switch0 have a latency of 10ms as well as the connection between the local server and switch0. Furthermore the connection between the web server and switch1 will have a latency of 10ms. All these connections have a latency of 10ms as in the context of the virtual network these are local connections. Finally the connection between switch1 and switch0 will have a latency of 40ms. The latency of 40ms was selected as 40ms is the latency incurred when streaming the DASH video from UCC servers. Therefore 40ms was representative of real world conditions when testing this DASH video content.

5.2 Testing Metrics

The quality of experience (QOE) metrics and the real world performance that corresponds to each of the QOE metrics are as follow:

- **P1203** A higher P1203 score corresponds to a video stream with a higher quality image. While lower P1203 score corresponds to a video stream with a lower overall image quality.

- **Clae** A lower Clae score implies a high average bitrate during the video stream with minimal switching between quality levels. A high Clae score implies a low average bitrate and a high degree of switching between image quality levels.

- **Duamu** A higher Duanmu score corresponds to a video with little stalling and a high average bitrate. This implies a video of higher image quality and low stalling rates. A lower Duanmu score implies a higher level of stalling and a lower bitrate.

- **Yin** A high Yin score corresponds to a high bitrate video stream that has minimal buffering at the start of the video stream and low levels of stalling throughout the video stream. This corresponds to a high image quality with low initial load times. A low Yin score corresponds to a low average bitrate and high levels of video stalling particularly at the start of the video stream.

- **Yu** A lower Yu score corresponds to a higher rate of stalling in the video stream. Stalling is when a video stream is forced to stop while waiting for the next video segment to arrive. A higher Yu score corresponds to a lower level of stalling in the video stream.

5.3 Adaptation Algorithm Selection

To determine the adaptation algorithm that would best exploit the Cooperative network each algorithm was tested in a variety of network conditions. Each of the test cases were as follows:

- All links are 2mb and 1 client
- All links are 2mb and 5 clients
- All links are 2mb and 10 client
- All links are 5mb and 1 client
- All links are 5mb and 5 clients
- All links are 5mb and 10 client
- All links are 10mb and 1 client

	p1203	clae	duanmu	yin	yu
avg all algorithms	1.877925159	0.160401389	35.64352103	-436.0235608	-2.065471627
avg conventional	2.030124444	0.17569	39.82741222	-880.1312789	-1.499376667
percentage improvement	0.081046513	0.095314705	0.117381534	1.018540643	-0.274075399

Figure 23: Results comparing avg of all adaptation algorithms to the best performing algorithm (Conventional)

- All links are 10mb and 5 clients
- All links are 10mb and 10 client

In each of the test cases each client streamed 40 seconds of video with each client producing a log of the QOE metrics for each segment downloaded. The average QOE metrics of all clients in each test case were then calculated to determine the adaptation algorithm with the greatest performance. The results can be seen in figure 23. The highest performing adaptation algorithm was the Conventional algorithm. In comparison to the other adaptation algorithms on average the Conventional algorithm produced a P1203 score 8% higher, a 11% higher Duanmu score and a 27% higher Yu score. The closest performing algorithm to the Conventional algorithm was the BBA2 algorithm which outperformed the conventional algorithm in the 10mb links single client test case. Despite this on average the Conventional algorithm outperformed BBA2 by 6% in P1203, 2% in Clae, 3% in Duanmu, 9 times better in yin and 6% better in yu. While the Conventional algorithm outperformed the other algorithms on average in most QOE metrics it falls behind the average in both Clae and Yin. While some algorithms do outperform it in these metrics they fall behind too often in other metrics to make the difference in both Clae and Yin scores significant enough to consider the algorithm as having better performance than the Conventional algorithm. As such the Conventional algorithm was chosen for use in the Cooperative model.

5.4 Single Video Stream Test

Now that the appropriate adaptation algorithm is selected it is time to begin our comparison of the Cooperative Peer-to-Peer Video Streaming model and the standard DASH model. For this we will be using the original network structure modifying the bandwidth of only the bottleneck link for each test case. This test will focus on the scenario where everyone in the network is streaming the same video content. With each client requesting the same video

content it will allow clients to effectively share content in the Cooperative network.This scenario is akin to many people in a watching a new Netflix release simultaneously This scenario suits the Cooperative network. In later tests we will explore the Cooperative models effectiveness when multiple videos are being streamed over the network. The test cases are as follows:

- Bottleneck link is 2mb, Single video is streamed, 1 client
- Bottleneck link is 2mb, Single video is streamed, 5 client
- Bottleneck link is 2mb, Single video is streamed, 10 client
- Bottleneck link is 5mb, Single video is streamed, 1 client
- Bottleneck link is 5mb, Single video is streamed, 5 client
- Bottleneck link is 5mb, Single video is streamed, 10 client
- Bottleneck link is 10mb, Single video is streamed, 1 client
- Bottleneck link is 10mb, Single video is streamed, 5 client
- Bottleneck link is 10mb, Single video is streamed, 10 client

These test cases were chosen as they explore many different scenarios. These test cases allow us to see the effectiveness of both the standard and cooperative model in the case where each client has surplus bandwidth, each client has adequate bandwidth and when each client has a lack of bandwidth.

The results of these test cases are as follows. In the cases where only one client is streaming on the network the Cooperative network was always outperformed by the Standard model. In P1203 it was outperformed by between 1 and 5 percent showing that the Cooperative network slightly lags behind in image quality in these cases. In both the Duanmu and Clae metrics the Cooperative model performs between 1 and 2 percent lower than the standard model showing the cooperative model has slightly higher bitrate with lower levels of stalling in these cases. Similarly the Standard DASH model outperforms the Cooperative model in the Yin by 6 to 50 percent and Yu by 2 to 3 percent showing again that the cooperative model incurs more stalling in these single client test cases. However these are the only cases where the Cooperative model is outperformed. In all other cases the Cooperative model outperforms the Standard model. Most notably in P1203 score the Standard

model is outperformed by 3 to 6 percent which is a good improvement in image quality while also significantly outperforming the Standard model in Yu score by 14 to 26 percent showing the Cooperative model can provide this higher image quality while stalling significantly less than the Standard model at a lower image quality in these test cases. Figure 23 shows the QOE results obtained in all test cases with a 5mb network link. From here on all graphs will show the percentage difference between the QOE results of the Standard model and the Cooperative model as shown in figure 25 which is based on the data shown in figure 24.

5mb 1 client 40 second Single Video	P1203	clae	duanmu	yin	yu
Standard Model	2.2362	0.1667	40.2591	-863.4295	0.5118
Cooperative Model	2.1252	0.1712	39.5352	-1368.1164	0.4116
5mb 5 clients 40 second Single Video	P1203	clae	duanmu	yin	yu
Standard Model	2.11416	0.1747	39.52062	-1327.67166	0.41338
Cooperative Model	2.22408	0.16866	40.17496	-886.7885	0.50034
5mb 10 clients 40 second Single Video	P1203	clae	duanmu	yin	yu
Standard Model	2.05204	0.177	39.20542	-1559.96295	0.36967
Cooperative Model	2.18351	0.17049	39.92754	-1072.79048	0.46672

Figure 24: The average QOE results of both the Cooperative and the Standard model in all test results with a 5mb link

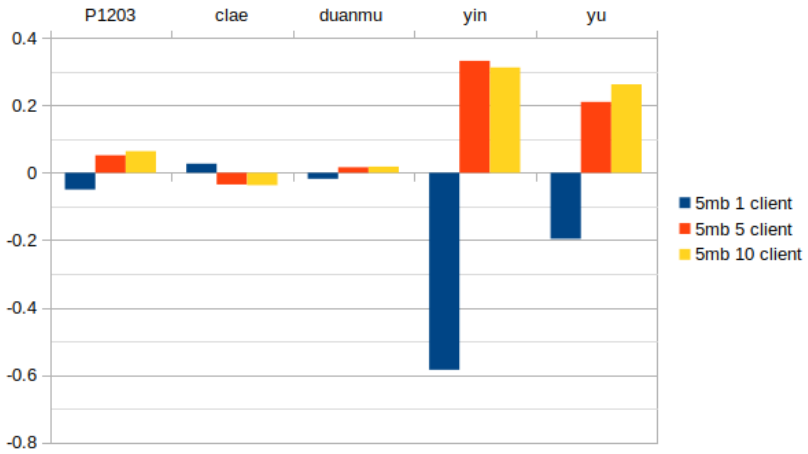


Figure 25: Percentage performance comparison between the average QOE of the Standard and Cooperative models in all cases with limited bandwidth of 5mb and all clients streaming a single video

Furthermore the in all test cases with more than a single video client, the Cooperative model's video clients had a lower minimum client QOE. When compared to the minimum QOE seen by the clients in the standard model the Cooperative model outperforms the standard model in all areas. Similarly in all cases with more than a single video client the highest average QOE of a video client in each test case was higher when using the Cooperative model. With both a higher minimum and maximum QOE it is clear to see that the Cooperative model outperforms the Standard model in all test cases with more than a single video client. While the Cooperative model falls behind in the test cases with a single video client it does manage to stay competitive. This lack of performance in single client test cases is due to the extra overhead of the Cooperative model and the inability for a single client to share content locally. While the Cooperative model stays competitive when outperformed the Standard model lags well behind the pace in all cases with five or more clients.

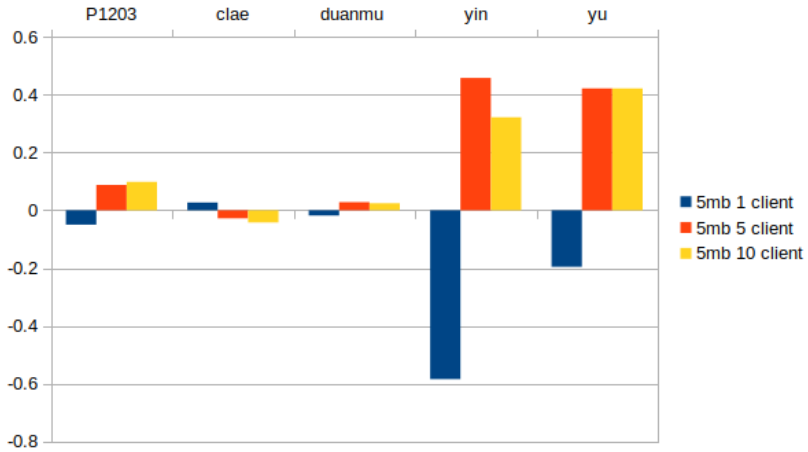


Figure 26: Percentage performance comparison between the minimum QOE of the Standard and Cooperative models in all cases with limited bandwidth of 5mb and all clients streaming a single video

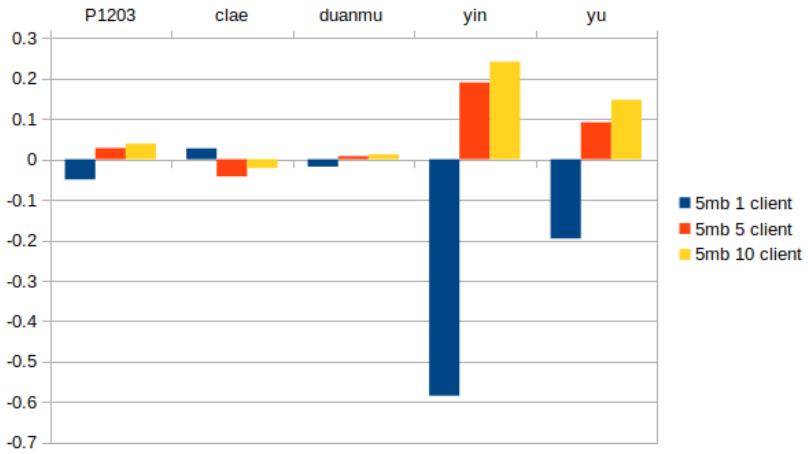


Figure 27: Percentage performance comparison between the maximum QOE of the Standard and Cooperative models in all cases with limited bandwidth of 5mb and all clients streaming a single video

Overall it is clear to see the advantages of the Cooperative network in the case where many users are requesting the same video content. While the Cooperative network is outperformed in single client test cases the difference in performance is quite small, this coupled with the Cooperative networks high performance in multiple client test cases makes it a great solution to improve the video streaming experience in a network where many clients are requesting the same video content with limited bandwidth.

5.5 Multiple Videos Stream Test

For this test we will be using the original network structure modifying the bandwidth of only the bottleneck link for each test case. This test will focus on the scenario in which there are four different video streams being randomly streamed over the network. Each video client will stream one of the four available video streams for 40 seconds and return the QOE logs of each segment downloaded. This scenario is more relevant to a real life scenario where many different videos may by streamed over the network. The test cases are the same as those used in the Single Video Stream test. The results of these tests are as follows. Similar to the Single Video Stream test the Cooperative model is outperformed on a QOE metrics. The Standard model outperforms the Cooperative model in Single client cases as follows, by 1%

Figure 28: Percentage performance comparison between the avg QOE of the Standard and Cooperative models in all cases with 1 client streaming 40 seconds of multiple videos

in P1203 2% in Clae, 0.003% in Duanmu, 6% in Yin and 4% in Yu. While outperformed in all QOE metrics the percentage the Cooperative model is outperformed relative to each metric is quite small.

In all test cases with five clients the Cooperative model still outperforms the Standard model although to a lesser extent this time outperforming the Standard model by between 1 and 2 percent in P1203 1 to 2 percent in Clae, 0.002 ti 0.008 percent in Duanmu, 5 to 11 percent in Yin and 3 to 10 percent in Yu. The most significant difference comes in the Yin and Yu score showing that while not providing significantly higher image quality than the Standard model the Cooperative model provides a video stream with far less stalling. As well as this the Cooperative model achieves a higher maximum QOE among its clients in all cases with more than one video client. The Cooperative model achieved a maximum QOE value outperforming the Standard model by the following, 3 to 4 percent in P1203, 2 to 6 percent in Duanmu 19 to 24 percent in Yin and 9 to 14 percent in Yu. Despite this higher maximum QOE the Cooperative networks minimum client QOE is lower than that of the Standard model by the following 2 to 5 percent in

P1203, 2 to 5 percent in Clae, 1 percent in Duanmu, 5 to 13 percent in Yin and 7 to 23 percent in Yu. The Cooperative models lower minimum and higher maximum QOE mean there is a wider spread in the user experience when using the Cooperative network while the Standard network appears to treat clients slightly more equally.

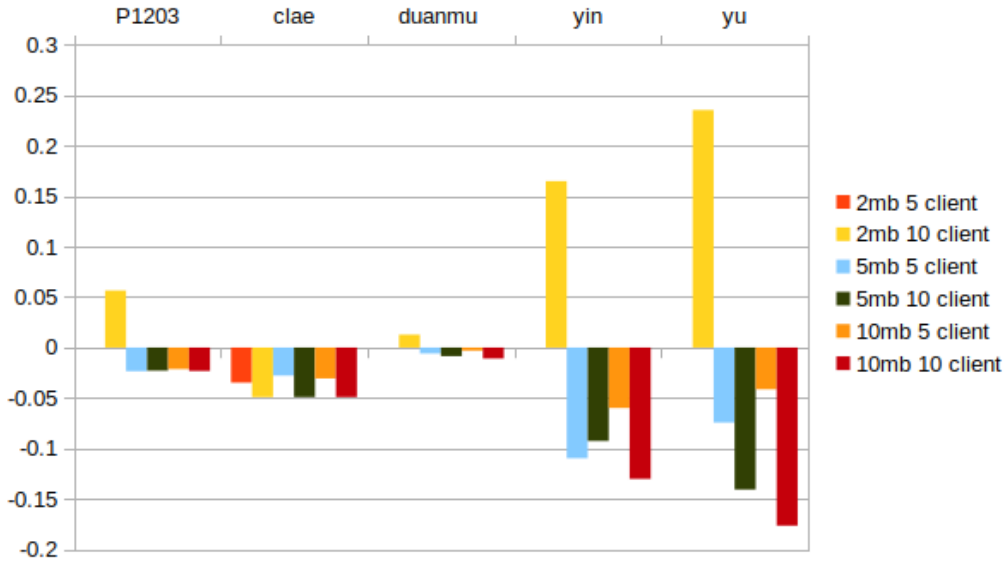


Figure 29: Percentage performance comparison between the minimum QOE of the Standard and Cooperative models in all cases with 5 and 10 clients streaming 40 seconds of multiple videos

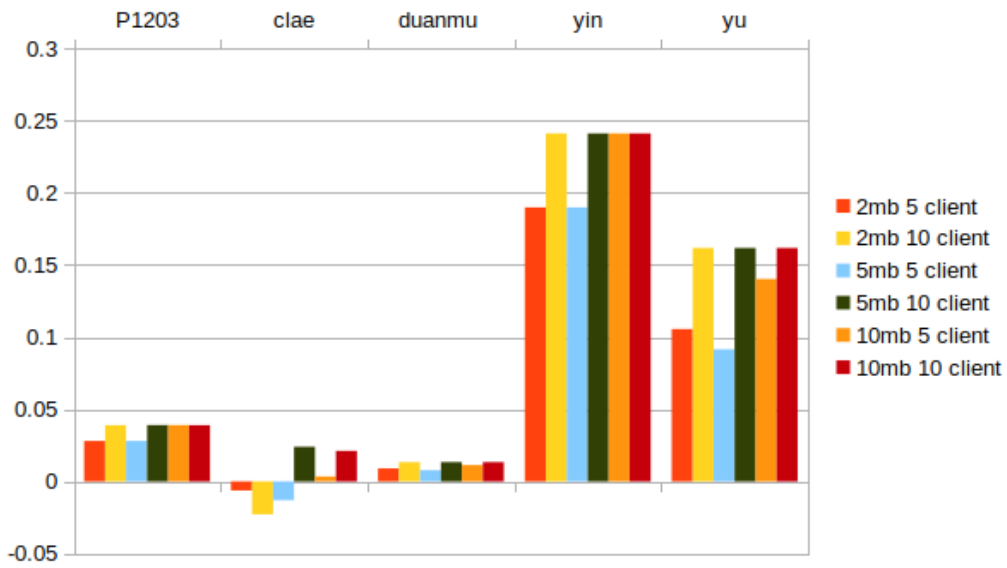


Figure 30: Percentage performance comparison between the maximum QOE of the Standard and Cooperative models in all cases with 5 and 10 clients streaming 40 seconds of multiple videos

The difference in performance becomes more apparent in the cases where bandwidth is more limited. In all test cases with ten clients the Cooperative network outperforms the Standard network by the following percentages, by between 3 and 6 percent in P1203, 2 to 3 percent in Clae, 1 percent in Duanmu, 14 to 26 percent in Yin and 13 to 24 percent in Yu. Again we can see the most significant difference in QOE comes from both the Yin and Yu scores showing that the Cooperative network is providing video streams that stall significantly less. However in contrast to the cases with 5 clients it is now providing streams with significantly less stalling and better image quality. This again demonstrates how the Cooperative network excels in scenarios where bandwidth is limited in the network.

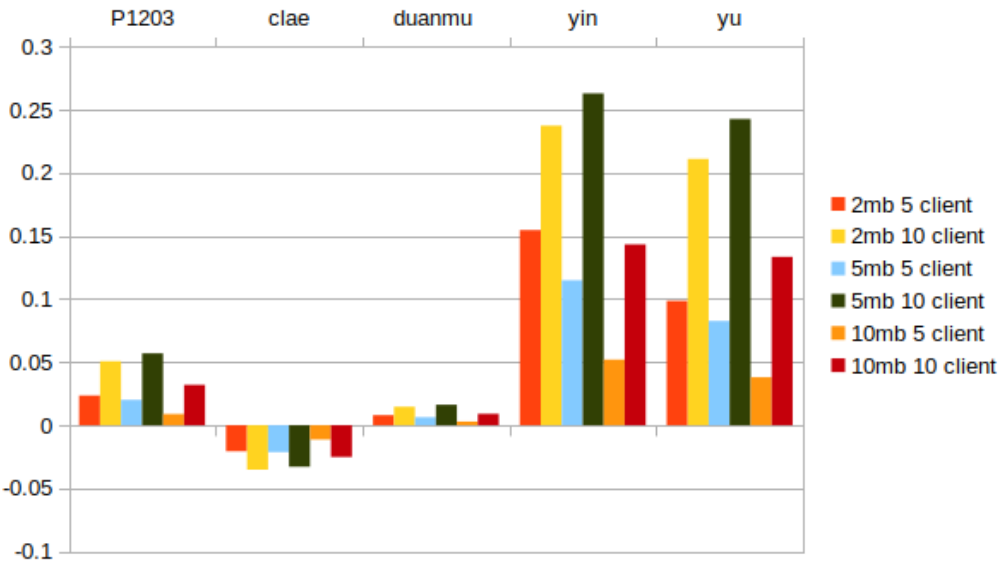


Figure 31: Percentage performance comparison between the avg QOE of the Standard and Cooperative models in all cases with 5 and 10 video clients streaming 40 seconds of multiple videos

Overall while the effectiveness of the Cooperative network is lessened when multiple videos are being streamed in the network a significant improvement in the amount of stalling occurring in the video streams can by had by utilising the Cooperative network. Furthermore in the case where bandwidth is limited the Cooperative network can also provide a higher image quality to the video streaming client. Despite the higher variance in the QOE provided the average experience of the video clients is improved using the Cooperative network.

5.6 Two Minute Video 10 Client Stream Test

This test will focus on the streaming of a longer segment of video with limited badnwidth available in the network. The standard virtual test network will be used. Each client in the network will download 120 seconds of video. Each client will stream the same video content. The test cases are as follows.

- Bottleneck link is 2mb, Single video is streamed, 10 client
- Bottleneck link is 5mb, Single video is streamed, 10 client

The results of the above test cases are as follows. In the test case with a network link of 5mb the Cooperative network significantly outperformed the Standard network in most metrics while falling behind equaling the Standard model in others. The Cooperative network outperformed the Standard network by the following, 20% in P1203, 22% in Duanmu, 183% in yin. The Cooperative model falls behind by 5% in Clae and 13% in yu this is due to the stalling in the network being roughly equal to the Standard model.

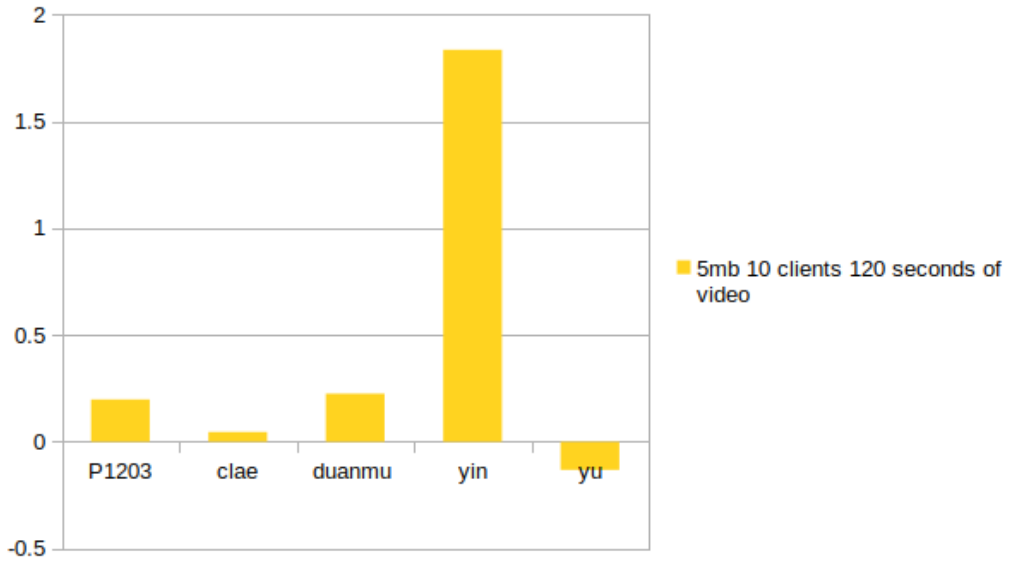


Figure 32: Percentage performance comparison between the avg QOE of the Standard and Cooperative models in the case with 10 clients on a 5mb link streaming 120 seconds of a single video

This is significant difference in the average QOE of the video streaming clients. Not only does the Cooperative network provide significantly higher image quality but also provides streams with similar levels of stalling to the Standard model. In line with the Cooperative networks average QOE performance in this test case both the Cooperative networks Minimum QOE and Maximum QOE significantly outperform the Standard network with 15% higher P1203 score minimum and 22% higher maximum P1203 score. This higher minimum and maximum QOE means that despite the Cooperative network having a higher variance between the QOE of clients the QOE of each client is higher than the Maximum QOE achieved by any video client

in the Standard network.

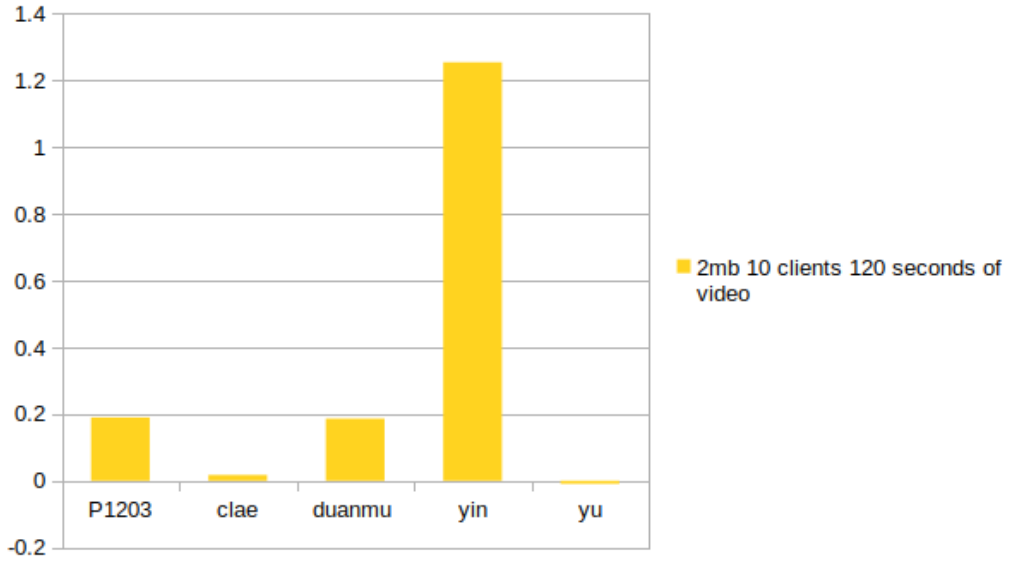


Figure 33: Percentage performance comparison between the avg QOE of the Standard and Cooperative models in the case with 10 clients on a 2mb link streaming 120 seconds of a single video

The case in much the same with regards the second test case with a 2mb network link.The Cooperative network outperformed the Standard network by the following, 19% in P1203, 2% in Clae, 18% in Duanmu, 125% in yin, 0% in Yu. However in comparison to the previous test case despite the Cooperative network providing significantly higher image quality it appears that in this case the Cooperative network stalls a similar amount to the standard network. This is illustrated by the similar average Yu score. Despite the similar yu score the Cooperative network still provides a higher overall QOE providing a minimum image quality that is equal to the maximum image quality achieved by the Standard network. This is illustrated by the Cooperative network achieving a minimum P1203 score of 2.20733 and the Standard network achieving a maximum P1203 score of 2.202.

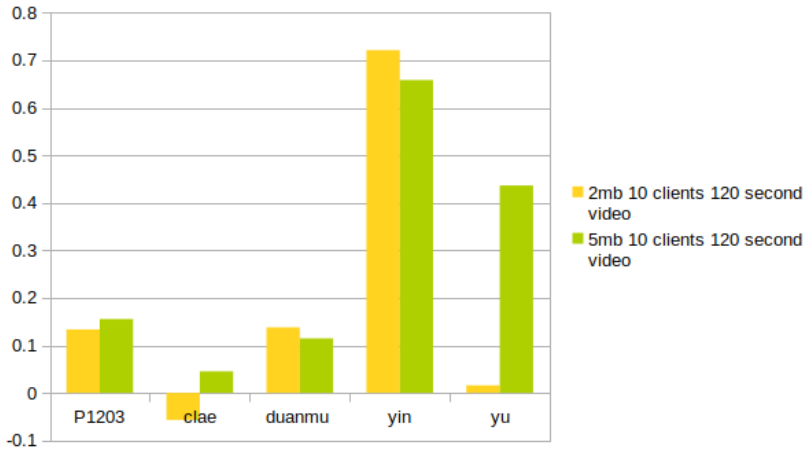


Figure 34: Percentage performance comparison between the minimum QOE of the Standard and Cooperative models in the case with 10 clients on a 2mb and 5mb link streaming 120 seconds of a single video

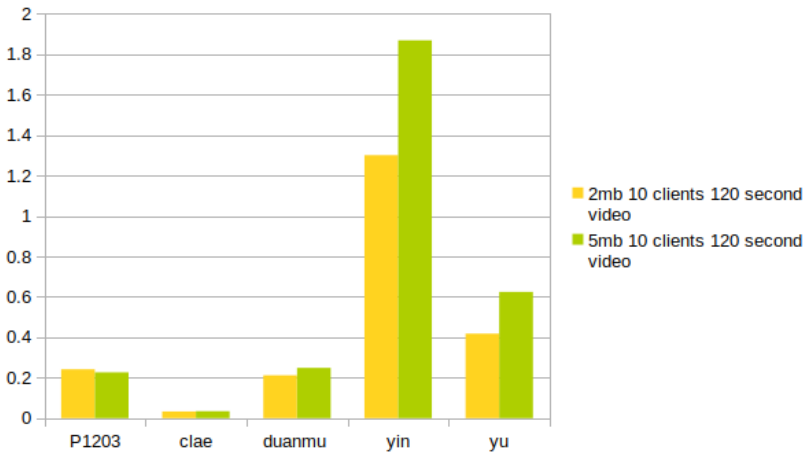


Figure 35: Percentage performance comparison between the maximum QOE of the Standard and Cooperative models in the case with 10 clients on a 2mb and 5mb link streaming 120 seconds of a single video

Overall it is clear that when faced with limited bandwidth and longer pieces of video content the Cooperative network provides a significant improvement in QOE over the Standard model.

5.7 Web Server Load

While the tests so far have focused on the QOE of the end user we will now look at the effect of using the Cooperative network on the Web Server. Specifically the reduction in the number of requests seen by the web server. Due to the fact that video clients can now share video content locally in the network this leads to a great reduction in the number of outgoing server requests.

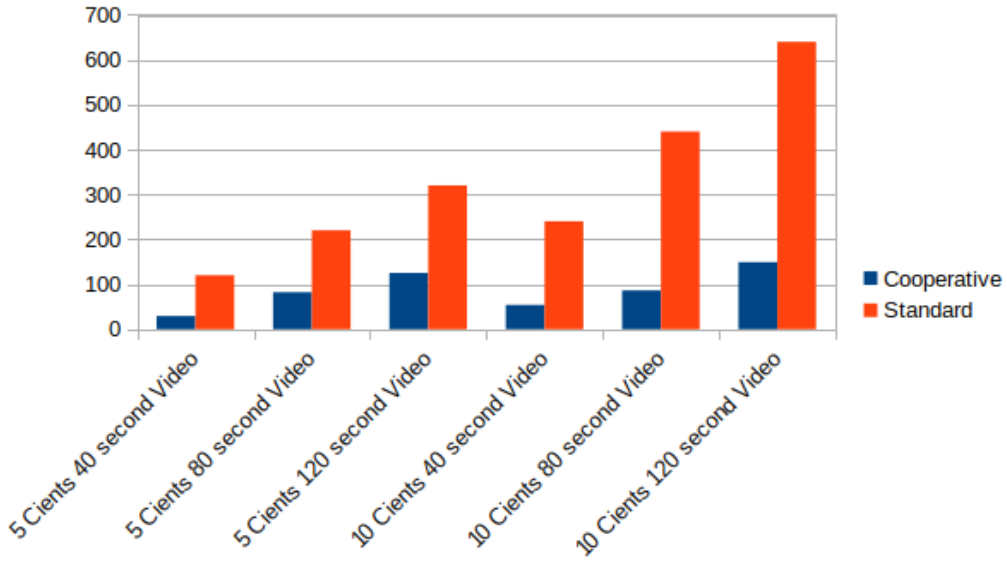


Figure 36: Graph of the number of web server requests received in 6 test cases comparing the Cooperative and Standard model

Figure 36 shows the comparison in the number of outgoing requests produced by the Cooperative Network and the Standard network. It is clear from figure 36 that the Cooperative network provides a substantial reduction in the number of outgoing web server requests. In all test cases with five clients the Cooperative network produced 63% less outgoing web server requests than the Standard network on average. Furthermore we see a larger difference in the volume of outgoing web server requests when regarding the test cases involving ten clients. The results for the test cases with ten clients were as follows. For ten clients downloading 40 seconds of video there was a 78.5% reduction in outgoing web server requests. For ten clients downloading 80 seconds of video there was a 80.5% reduction in outgoing web server

requests and for ten clients downloading 120 seconds of video there was a 77% reduction in the number outgoing web server requests. This decrease in requests takes a load off of the external network allowing other internet users to utilise bandwidth that was otherwise required for video streaming. This reduction in requests shows that the overall load on the network can be reduced using the Cooperative network while the above QOE results show it will have a no impact or event a positive impact of video client QOE.

5.8 Final Analysis

From the above tests we can see that the Cooperative model is hindered in all single video client applications. This lack of performance in single video client applications is caused by the addition of latency in the video client caused by the overhead in the Cooperative models framework. While the Cooperative model does hinder single client use cases the level at which the QOE of the video client in these single client use cases is hindered is relatively low making it an acceptable trade off for the extra performance the Cooperative model offers in multi video client use cases.In multi client applications the Cooperative model equals or outperforms the Standard model in the QOE achieved by the video clients while also also reducing the overall bandwidth used in the network. While the QOE of video clients is maintained or improved in the Cooperative network it is worth noting that the Cooperative network still suffers from stalling in these multi client applications similar to the Standard network. This occasional stalling occurs as a result of a peer in the network receiving too many requests for video content simultaneously and being unable to immediately serve each client. A Cooperative network with more efficient load balancing may be able to improve or even negate this issue. Overall we see that the Cooperative model outperforms the Standard model in all multi video client use cases while reducing the bandwidth used in the network. Furthermore the Cooperative model offers adequate performance in single client applications, although not to the same level as the Standard model, making the Cooperative model a viable solution for reducing the bandwidth needed to stream video in the network while maintaining video client QOE.

6 Conclusion

This project set out to create a Cooperative Peer-to-Peer Video Streaming network capable of reducing the bandwidth required to stream video content while maintaining the QOE of video clients in the network. This project utilised goDASH, a DASH video player implemented in Golang, as the video player for each video client in the Cooperative network. This project utilised a hybrid Peer-to-Peer structure to allow peers to share previously downloaded video content over the local network with other peers. Hashicorps Consul server was used as the local server in the Cooperative network . Consul's Key-Value system tracked what video content is present on the network as well as what peer is hosting the corresponding video content. When new video content becomes available on a peer in the network that peer updates Consul with a new Key-Value entry corresponding to the new video content available from that peer. The project utilises GRPC, a general remote procedure call framework, to implement efficient communication between peers in the network allowing peers to request video content from known peers on the network. When a peer searches for a segment of video content the peer first searches its known peers by sending a search request, using GRPC, to all known peers. If a known peers is hosting the requested video content that peer will return the URL of the requested video content for goDASH to stream. If the video content is not present among the peers known peers it then requests the video content from the local Consul server. If the video content is present anywhere on the network Consul will return the address of the peer hosting the requested content. The peer then requests the URL of the desired video content from the address returned by Consul. If the video content is not present on the network goDASH will stream the desired video content from the internet as normal. This is a summary of the Cooperative Peer-to-Peer Video Streaming model created during this project.

The Cooperative model was then tested using a modified version of go-DASHbed a goDASH testing framework that simulates real world network conditions using Mininet. Through testing it was found that the Cooperative model slightly hinders the QOE of video clients in single client test cases. This slight lowering of video client QOE in single video client cases is caused by additional latency created by the Cooperative model framework. However this reduction in QOE is relatively small and the Cooperative model still provides adequate QOE performance when compared with the Standard DASH

model. In single client applications on average the Cooperative model produces the following percentage lower than the Standard DASH model , 1% lower P1203 score, 2% lower Clae score, 0.003% lower in Duanmu, 6% lower in yin score and 4% lower Yu score. Where the Cooperative model shines is in multi user applications with limited network bandwidth. In multi video client test cases the Cooperative model maintained and often improved the QOE of video clients when compared with the Standard DASH model in the same test case. In all cases with five or more clients the Cooperative model outperforms the Standard model. Most notably in P1203 score the Standard model is out-performed by 3 to 6 percent which is a good improvement in image quality while also significantly outperforming the Standard model in Yu score by 14 to 26 percent showing the Cooperative model can provide this higher image quality while stalling significantly less then the Standard model at a lower image quality in these test cases. Further more the Cooperative model maintained these QOE levels while significantly reducing the number of outgoing web server requests, reducing the number of outgoing requests by as much as 80% in some test cases.

The ability of the Cooperative network to maintain or even improve the QOE of the video clients while significantly reducing the volume of outgoing traffic on the network make the Cooperative video streaming models like the one presented in this project a viable solution in reducing the amount of bandwidth needed to stream video. In an age where bandwidth is a precious commodity it is important we fully utilise the resources available to us in order to maintain and even improve the QOE of the internet as a whole.

References

[1] Sandvine. The mobile internetphenomena report. Technical report, Sandvine, 2020.

[2] Carter Lewis. Web could collapse as video demand soars. *The Daily Telegraph London*, 2008. Accessed: 2020-03-23.

[3] Sandvine. The global internet phenomena report. Technical report, Sandvine, 2011.

[4] Thomas Stockhammer. Dynamic adaptive streaming over http– standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144, 2011.

[5] Dilip Kumar Krishnappa, Divyashri Bhat, and Michael Zink. Dashing youtube: An analysis of using dash in youtube video service. In *38th Annual IEEE Conference on Local Computer Networks*, pages 407–415. IEEE, 2013.

[6] Sandvine. The global internet phenomena report. Technical report, Sandvine, 2019.

[7] RTE. Covid-19: Netflix to reduce streaming quality in europe for 30 days. *RTE*, 2020.

[8] Gang Peng. Cdn: Content distribution network. *arXiv preprint cs/0411069*, 2004.

[9] Min Yang and Yuanyuan Yang. An efficient hybrid peer-to-peer system for distributed data sharing. *IEEE Transactions on computers*, 59(9):1158–1171, 2009.

[10] Protobuf. https://developers.google.com/protocol-buffers. Accessed: 2020-03-22.

[11] Janakiram MSV. grpc - the protocol of microservices joins the cloud native computing foundation. https://www.forbes.com/sites/janakirammsv/2017/03/01/grpc-the-protocol-of-microservices-joins-the-cloud-native-computing-foundation/#3cde69e14b0a. Accessed: 2020-04-01.

[12] International Telecommunication Union. H.264 : Advanced video coding for generic audiovisual services. https://www.itu.int/rec/T-REC-H.264. Accessed: 2020-03-30.

[13] BitMovin. 2019 video developer report. Technical report, BitMovin, 2019.

[14] Rafael Huysegems, Jeroen Van Der Hooft, Tom Bostoen, Patrice Rondao Alface, Stefano Petrangeli, Tim Wauters, and Filip De Turck. Http/2-based methods to improve the live experience of adaptive streaming. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 541–550, 2015.

[15] SoftwareX Elsevier. godash. submission under review.

[16] Zhengfang Duanmu, Abdul Rehman, and Zhou Wang. A quality-of-experience database for adaptive video streaming. *IEEE Transactions on Broadcasting*, 64(2):474–487, 2018.

[17] Stefano Petrangeli, Jeroen Famaey, Maxim Claeys, Steven Latré, and Filip De Turck. Qoe-driven rate adaptation heuristic for fair adaptive video streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 12(2):1–24, 2015.

[18] Li Yu, Tammam Tillo, and Jimin Xiao. Qoe-driven dynamic adaptive video streaming strategy with future information. *IEEE Transactions on Broadcasting*, 63(3):523–534, 2017.

[19] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.

[20] Alexander Raake, Marie-Neige Garcia, Werner Robitza, Peter List, Steve Göring, and Bernhard Feiten. A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1. In *Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, Erfurt, May 2017. IEEE.

[21] Werner Robitza, Steve Göring, Alexander Raake, David Lindegren, Gunnar Heikkilä, Jörgen Gustafsson, Peter List, Bernhard Feiten, Ulf

Wüstenhagen, Marie-Neige Garcia, et al. Http adaptive streaming qoe estimation with itu-t rec. p. 1203: open databases and software. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 466–471, 2018.

[22] Darijo Raca, Jason J Quinlan, Ahmed H Zahran, and Cormac J Sreenan. Beyond throughput: a 4g lte dataset with channel and context metrics. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 460–465, 2018.

[23] Jason J Quinlan, Ahmed H Zahran, and Cormac J Sreenan. Datasets for avc (h. 264) and hevc (h. 265) evaluation of dynamic adaptive streaming over http (dash). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–6, 2016.

[24] sas.co.uk. What is network latency (and how do you use a latency calculator to calculate throughput)? https://www.sas.co.uk/blog/what-is-network-latency-how-do-you-use-a-latency-calculator-to-calculate-throughput. Accessed: 2020-04-16.

Appendices

A Code for StartListening function

The below code show the initialisation of the GRPC server of a peer in the cooperative network. This server serves all search requests received by the peer and returns the appropriate URL of the requested content is present on the peer.

```
//Start the node server listening
func (n *NodeUrl) StartListening() {
  lis, err := net.Listen("tcp", n.Addr)
  fmt.Printf("GRPC Server Listening on %v\n", n.Addr)
  if err != nil {
    log.Fatalf("failed to start listening %v", err)
  }

  _n := grpc.NewServer()

  pb.RegisterP2PServiceServer(_n, n)

  reflection.Register(_n)
  defer _n.Stop()
  if err := _n.Serve(lis); err != nil {
    fmt.Println("server failed")
    log.Fatalf("Server Failed to serve")
  }

}


```

Listing 10: StartListening function

B Code for ContentServerStart function

The below code shows the initialisation of the content server of a peer in the Cooperative network. The file location of the content to be servered and the port the content will be served on is passed to the function. The function then creates a server to serve video content on the provided port from the given file location.

```
    func (n *NodeUrl) ContentServerStart(location string,
   port string) {
  server := http.NewServeMux()

  //handlers that serve the home html file when called
  fs := http.FileServer(http.Dir(location))

  //handles paths by serving correct files
  server.Handle("/", fs)

  //logs that server is Listening
  log.Printf("Listening... %v\n", location)

  //starts server
  http.ListenAndServe(n.IP.String()+port, server)
}

```

Listing 11: ContentServerStart function

C Code for Search function

The below code is the Search function used by peers in the Cooperative network in order to find video content on the local network. The function takes in a string of the requested video contents URL. The function first sends search requests to all known peer. If the video content is present on a known peer the peer will return the URL of the content for goDASH to stream. If the video content is not present among the known peers the peer will then query Consul. Consul will return the address of the device hosting the video content if the video content is present anywhere on the network. The peer then requests the URL of the desired video content from the address Consul returned. If the video content is not present on the network the originally requested URL is returned and goDASH streams the video content from the internet as normal.

```
//search network for a given url
func (n *NodeUrl) Search(url string) string {
  start := time.Now()
  notFound := true
  l := strings.Split(url, "/")
  location := l[len(l)-1]

```

```
  key := hlpr.HashSha(url)

  //if desired content is not in current clients
  //search clients of clients
  if len(n.Clients[key]) != 0 {
    //if current client is known to have correct content from previous requests
    contentServer, err := n.GetContentServerAddress(n.Clients[key])
    if err != nil {
      fmt.Printf("Error Local Client: %v\n", err)
    }else {
      url = "http://" + contentServer + "/" + location + "::localclient"
      notFound = false
    }
  }
  //loop over all known nodes
  for _, client := range n.Clients {

    //establish connection to client and check for content
    conn, err := grpc.Dial(client, grpc.WithInsecure())
    if err != nil {
      continue
      log.Fatalf("Did not connect to server : %v\n", err)
    }

    defer conn.Close()
    c := pb.NewP2PServiceClient(conn)
    //GRPC call to check clients for content
    downloadAddress, err := c.CheckClients(context.Background(), &pb.CheckRequest{Address: n.Addr, Target: key})
    fmt.Println("check 6")
    if err != nil {
      fmt.Println("rpc error in checkclients")
      continue
    }

    if downloadAddress.Addr != "nil" {
      //add relevant client to client list
      n.Clients[key] = downloadAddress.Addr

      //get content server address for Url for download
      contentServer, err := n.GetContentServerAddress(downloadAddress.Addr)
```

```
48        if err != nil {
49          fmt.Printf("Error Client of client : %v\n", err)
50          break
51        }
52        url = "http://" + contentServer + "/" + location+"::
    clients"
53        fmt.Println("Download from currently known clients")
54
55        notFound = false
56        break
57      }
58    }
59
60
61    //in Case not currently known Locally consult Consul Server
62    if notFound {
63      fmt.Println("consul loop")
64      kvpairs, _, err := n.SDKV.List(key, nil)
65
66      if err != nil {
67        fmt.Printf("Consul Error Download from internet\n")
68        return url
69      } else {
70        //Loop Key Value pair matches query
71        //randomly shuffle key value pairs
72        for i := 1; i < len(kvpairs); i++ {
73          r := rand.Intn(i + 1)
74          if i != r {
75            kvpairs[r], kvpairs[i] = kvpairs[i], kvpairs[r]
76          }
77        }
78        for _, kventry := range kvpairs {
79          //Check key isnt this node
80          if kventry.Key[0:len(key)] == key && kventry.Key !=
    key+n.Addr {
81            n.Clients[kventry.Key[0:len(key)]] = string(kventry
    .Value)
82
83            contentServer, err := n.GetContentServerAddress(
    string(kventry.Value))
84            if err != nil {
85              continue
86              fmt.Println("KV ERROR")
87            } else {
```

```
88              url = "http://" + contentServer + "/" + location
    +"::consul"
89              notFound = false
90              break
91            }
92          }
93        }
94      }
95    }
96    fmt.Printf("Returned URL : %v\n", url)
97
98    return url+"::"+time.Since(start).String()
99  }
100
```

Listing 12: Search function

D Test Result tables

D.1 Average QOE in Single video of 40 second length Test

Below are the percentage the Cooperative models average video client QOE performs better or worse in comparison to the average video client QOE of the Standard model in the same test cases.

Comparison of porformance	P1203	clae	duanmu	yin	yu
2mb 1 client	-0.013818084	0.018596281	-0.003417861	0.067634358	-0.036928488
2mb 5 client	0.032563597	-0.034167524	0.011689561	-0.2033381	0.146921686
2mb 10 client	0.056535209	-0.034615385	0.017025749	-0.282947206	0.24321974
	P1203	clae	duanmu	yin	yu
5mb 1 client	-0.049637778	0.026994601	-0.017981028	-0.584514312	-0.195779601
5mb 5 client	0.051992281	-0.034573555	0.016556926	0.332072434	0.210363346
5mb 10 client	0.064067952	-0.036779661	0.018418882	0.312297462	0.262531447
10mb 1 client	-0.013818084	0.018596281	-0.003417861	0.067634358	-0.036928488
10mb 5 client	0.044064204	-0.03335618	0.0144234	-0.277091447	0.183244052
10mb 10 client	0.060834243	-0.038361582	0.017710036	-0.287430993	0.246827051

Figure 37: Percentage performance comparison between the average QOE of the Standard and Cooperative models in all cases streaming 40 seconds of single video

D.2 Minimum QOE in Single video of 40 second length Test

Below are the percentage the Cooperative models minimum video client QOE performs better or worse in comparison to the minimum video client QOE of the Standard model in the same test cases.

Comparison of porformance	P1203	clae	duanmu	yin	yu
2mb 1 client	-0.013818084	0.018596281	-0.003417861	0.067634358	-0.036928488
2mb 5 client	0.018445448	-0.034742328	0.007512169	-0.108153272	0.112024666
2mb 10 client	0.040018423	-0.049058756	0.009138351	-0.116067481	0.158049738
	P1203	clae	duanmu	yin	yu
5mb 1 client	-0.049637778	0.026994601	-0.017981028	-0.584514312	-0.195779601
5mb 5 client	0.087801509	-0.027793862	0.028147431	0.458028194	0.421690222
5mb 10 client	0.098504837	-0.042213349	0.024237688	0.321840566	0.421662125
10mb 1 client	-0.013818084	0.018596281	-0.003417861	0.067634358	-0.036928488
10mb 5 client	0.071407201	-0.037528868	0.021957482	-0.325129006	0.335217133
10mb 10 client	0.074255674	-0.042213349	0.018643911	-0.260687786	0.295969773

Figure 38: Percentage performance comparison between the minimum QOE of the Standard and Cooperative models in all cases streaming 40 seconds of single video

D.3 Maximum QOE in Single video of 40 second length Test

Below are the percentage the Cooperative models maximum video client QOE performs better or worse in comparison to the maximum video client QOE of the Standard model in the same test cases.

Comparison of performance	P1203	clae	duanmu	yin	yu
2mb 1 client	-0.013818084	0.018596281	-0.003417861	0.067634358	-0.036928488
2mb 5 client	0.028038242	-0.024943311	0.008984827	-0.189960374	0.105638367
2mb 10 client	0.039070848	-0.022333892	0.013368405	-0.241242467	0.161861521
5mb 1 client	-0.049637778	0.026994601	-0.017981028	0.584514312	-0.195779601
5mb 5 client	0.028038242	-0.042301184	0.007769307	-0.189960374	0.091596457
5mb 10 client	0.039070848	-0.021240917	0.012147604	-0.241242467	0.147105562
10mb 1 client	-0.013818084	0.018596281	-0.003417861	0.067634358	-0.036928488
10mb 5 client	0.035894586	-0.024943311	0.011883488	-0.241242467	0.143176234
10mb 10 client	0.039070848	-0.027512633	0.012147604	-0.241242467	0.147105562

Figure 39: Percentage performance comparison between the maximum QOE of the Standard and Cooperative models in all cases streaming 40 seconds of single video

D.4 Average QOE in single video of 120 second length Test

Below are the percentage the Cooperative models average video client QOE performs better or worse in comparison to the average video client QOE of the Standard model in the same test cases.

	P1203	clae	duanmu	yin	yu
Staadard 5mb 10 client 2 min video	2.09642	0.209423333	41.73786667	16843.3914	-7.22113
Cooperative 5mb 10 client 2 min video	2.51091	0.219036667	51.12994667	47740.75693	-8.17056
5mb 10 clients 120 seconds of video	0.197713244	0.045903831	0.225025397	1.834390995	-0.131479422
	P1203	clae	duanmu	yin	yu
Stnadard 2mb 10 client 2 min video	2.03419	0.21021	42.40162	19861.7497	-8.795
Cooperative 2mb 10 client 2 min video	2.42036	0.213856667	50.29955	44770.54796	-8.875323333
2mb 10 clients 120 seconds of video	0.18983969	0.017347732	0.186264817	1.254108959	-0.009132841

Figure 40: Percentage performance comparison between the average QOE of the Standard and Cooperative models in all cases streaming 120 seconds of single video

D.5 Minimum QOE in Single video of 120 second length Test

Below are the percentage the Cooperative models minimum video client QOE performs better or worse in comparison to the minimum video client QOE

of the Standard model in the same test cases.

	P1203	clae	duanmu	yin	yu
Standard 2mb 10 clients	1.947466667	0.2029	41.466	16981.30537	-11.11656667
Cooperative 2mb 10 clients	2.207333333	0.191366667	47.17836667	29226.08267	-11.2926
2mb 10 clients 120 second video	0.133438313	-0.056842451	0.137760253	0.721073971	0.015835225
Standard 5mb 10 clients	1.987566667	0.200466667	41.0199	15645.28253	-7.699366667
Cooperative 5mb 10 clients	2.296133333	0.209533333	45.71973333	25941.8633	-11.0604
5mb 10 clients 120 second video	0.155248461	0.045227802	0.114574471	0.658126866	0.436533741

Figure 41: Percentage performance comparison between the minimum QOE of the Standard and Cooperative models in all cases streaming 120 seconds of single video

D.6 Maximum QOE in Single video of 120 second length Test

Below are the percentage the Cooperative models maximum video client QOE performs better or worse in comparison to the maximum video client QOE of the Standard model in the same test cases.

	P1203	clae	duanmu	yin	yu
Standard 2mb 10 clients	2.202233333	0.2184	43.66803333	22579.6352	-6.895133333
Cooperative 2mb 10 clients	2.7348	0.225666667	52.9255	51940.32457	-4.016333333
2mb 10 clients 120 second video	0.241830263	0.033272283	0.211996418	1.30031726	0.417511868
Standard 5mb 10 clients	2.252466667	0.2184	42.62566667	19009.78073	-6.802066667
Cooperative 5mb 10 clients	2.7619	0.225933333	53.22043333	54520.20203	-2.556566667
5mb 10 clients 120 second video	0.226166869	0.034493284	0.248553688	1.868007938	0.624148543

Figure 42: Percentage performance comparison between the maximum QOE of the Standard and Cooperative models in all cases streaming 120 seconds of single video

D.7 Average QOE in multiple videos of 40 second length Test

Below are the percentage the Cooperative models average video client QOE performs better or worse in comparison to the average video client QOE of the Standard model in the same test cases.

Comparison of porformance	P1203	clae	duanmu	yin	yu
2mb 1 client	-0.013773366	0.018596281	-0.003420345	-0.067934904	-0.036928488
2mb 5 client	0.023579679	-0.020613834	0.008001734	0.154634499	0.09855416
2mb 10 client	0.050712937	-0.035165701	0.014725298	0.23740374	0.211034067
5mb 1 client	-0.013818084	0.018596281	-0.003417861	-0.067634358	-0.036928488
5mb 5 client	0.020181406	-0.021262003	0.00637184	0.114862798	0.082654586
5mb 10 client	0.057011081	-0.032788732	0.016300623	0.262981279	0.242617628
10mb 1 client	-0.013818084	0.018596281	-0.003417861	-0.067634358	-0.036928488
10mb 5 client	0.00889928	-0.011212815	0.002931914	0.052021526	0.03808752
10mb 10 client	0.032063179	-0.025161835	0.009076123	0.143308294	0.133447881

Figure 43: Percentage performance comparison between the average QOE of the Standard and Cooperative models in all cases streaming 40 seconds of multiple videos

D.8 Minimum QOE in multiple videos of 40 second length Test

Below are the percentage the Cooperative models minimum video client QOE performs better or worse in comparison to the minimum video client QOE of the Standard model in the same test cases.

Comparison of porformance	P1203	clae	duanmu	yin	yu
2mb 1 client	0	0	0	0	0
2mb 5 client	0	-0.034742328	0	0	0
2mb 10 client	0.056459579	-0.049058756	0.012736691	0.164850145	0.235253054
5mb 1 client	-0.013818084	0.018596281	-0.003417861	-0.067634358	-0.036928488
5mb 5 client	-0.023458574	-0.027793862	-0.00597843	-0.109554672	-0.07423231
5mb 10 client	-0.022832475	-0.049058756	-0.008418816	-0.092478236	-0.140568099
10mb 1 client	-0.013818084	0.018596281	-0.003417861	-0.067634358	-0.036928488
10mb 5 client	-0.021289919	-0.030600462	-0.003102584	-0.059909347	-0.041388518
10mb 10 client	-0.023238887	-0.049058756	-0.010795655	-0.130044163	-0.176265271

Figure 44: Percentage performance comparison between the minimum QOE of the Standard and Cooperative models in all cases streaming 40 seconds of multiple videos

D.9 Maximum QOE in multiple video of 40 second length Test

Below are the percentage the Cooperative models maximum video client QOE performs better or worse in comparison to the maximum video client QOE of the Standard model in the same test cases.

Comparison of porformance	P1203	clae	duanmu	yin	yu
2mb 1 client	-0.013773366	0.018596281	-0.003420345	-0.067934904	-0.036928488
2mb 5 client	0.028038242	-0.006239365	0.008984827	0.189960374	0.105638367
2mb 10 client	0.038974311	-0.022803115	0.013368405	0.241242467	0.161861521
5mb 1 client	-0.013818084	0.018596281	-0.003417861	-0.067634358	-0.036928488
5mb 5 client	0.028038242	-0.012972363	0.007769307	0.189960374	0.091596457
5mb 10 client	0.039070848	0.023982153	0.013368405	0.241242467	0.161861521
10mb 1 client	-0.013818084	0.018596281	-0.003417861	-0.067634358	-0.036928488
10mb 5 client	0.039070848	0.003422704	0.011343172	0.241242467	0.140374633
10mb 10 client	0.038974311	0.021134594	0.013368405	0.241242467	0.161861521

Figure 45: Percentage performance comparison between the maximum QOE of the Standard and Cooperative models in all cases streaming 40 seconds of multiple videos