AALBORG UNIVERSITY
STUDENT REPORT

Innovative Communication Technologies And Entrepreneurship
Aalborg University Copenhagen

Title:
Reliable and low-latency QUIC-based framework for OTT media delivery

Theme:
Master Project

Project Period:
01/03/2021 - 03/06/2021

Project Group:
Group 2.7

Participant(s):
Jonas K. B. Hansen - 20163811

Supervisor(s):
Cedomir Stefanovic (internal)
Allan Hammershøj (external)

Page Numbers: 49

Date of Completion:
June 3, 2021

Abstract:

As the audience for live streaming big events is growing, resolution requirements increasing and low latency in high demand, broadcasters are compelled to look for innovation on how to efficiently stream video, as the cost for distribution is rising. This project presents a solution to solve the problem using QUIC with multicast integration that works by keeping the convenient features of the existing QUIC protocol, with built-in security, reliability and the retransmission features on unicast. The solution is utilizing RLNC encoding to ensure reliable transmission on any IP multicast supported network, while transmitting the video content on a multicast stream, and using unicast to do packet acknowledgement, retransmission and initialization. A system architecture where the unicast is taking place over the public internet, while multicasting over a managed network, connected to setup boxes at customer premises, or in gateways close to the end clients. This results in having the last mile to be in standard unicast HTTP/2 connection, so all video playback devices with an internet connection, can enjoy a low latency high quality live stream.

Abstract

As the audience for live streaming big events is growing, resolution requirements increasing and low latency in high demand, broadcasters are compelled to look for innovation on how to efficiently stream video, as the cost for distribution is rising. This project presents a solution to solve the problem using QUIC with multicast integration that works by keeping the convenient features of the existing QUIC protocol, with built-in security, reliability and the retransmission features on unicast. The solution is utilizing RLNC encoding to ensure reliable transmission on any IP multicast supported network, while transmitting the video content on a multicast stream, and using unicast to do packet acknowledgement, retransmission and initialization. A system architecture where the unicast is taking place over the public internet, while multicasting over a managed network, connected to setup boxes at customer premises, or in gateways close to the end clients. This results in having the last mile to be in standard unicast HTTP/2 connection, so all video playback devices with an internet connection, can enjoy a low latency high quality live stream.

Contents

Chapter 1

Introduction

This project revolves around enabling efficient and low-latency media streaming over any internet protocol(IP) capable network. By implementing a custom version of the protocol QUIC, that enables it to work in two parallel streams. Firstly a standard QUIC unicast stream over the public internet to do packet repair, initial requests and security setup. Then another stream to handle multicast traffic over a managed network such as satelite, terrestrial, or even a 5g mobile network.

As multimedia applications with real-time video streaming, accounts for the majority of traffic on the internet [6]. Content Delivery Networks (CDNs) around the world are charging massive fees for transporting all of this traffic, and a typical edge server from Akamai can experiance up to 60 million request every second [15]. The main goal of this project is to facilitate a framework that enables large-scale video streaming with many viewers with low latency and high quality, with a smaller bandwidth footprint. Furthermore, the project investigates the integration of random linear network coding solution (RLNC) in the protocol stack, ensuring reliability and low signaling overhead, supporting unicast, multicast and broadcast media delivery, and enabling seamless transition from unicast to multicast delivery modes. The particular focus will be on media delivery over satellite networks in an all-IP environment. To deliver the video for the end user, the solution proposed is having a gateway, close by, so the last mile of the video stream is delivered with standard HTTP/2 unicast connection. The report presents a preliminary of knowledge to establish a general understanding of the main concept used in the analysis, and finally implementation of the solution. A big part of it is also to test and evaluate the solution with different configuration, and types of network. In the end the section *Section 6 Discussion and Future Work* will elaborate on the project results, discussing project results and its future improvements.

1.1 Background

In an era of video on demand services, the increase of video streaming traffic is over 70 percent of all Internet traffic [20]. This counts both live streaming and linear media content, although the popularity of on-demand viewing is increasing rapidly, the potential audience for live and linear content increases every year [5]. The Super Bowl in 2020, was watched by 99.9 million viewers [1], of that nearly 7.4 million of viewers tuned in through various Internet streaming services, an increase of 23% compared to 2019 [5]. The potential audience for streaming events like these is massive, but with the big viewer numbers, the scaling comes with a huge cost of broadcasting. Cellular technologies are increasingly dominating that space, and the companies behind those have sufficient capital to continue the expansion [5], inviting the broadcasters to look for innovation in how to efficiently deliver content. There is a need for solutions that can work better given the scarcity of the available bandwidth.

Amongst the COVID-19 pandemic, users of video streaming services like Netflix and YouTube experienced for the first time that the video quality was reduced for all viewers in Europe. Because of the total load on continents internet infrastructure due to increased home usage of the services [22]. This could result in digital industries being heavily throttled by high costs of delivery [5]. The audiences tuning in to Internet streaming sessions is scary, and increasing. Even if existing CDNs were capable of distributing such loads, the costs at that scale would have great financial cost for any broadcaster [5]. By reducing transmission costs associated with the content delivery at scale, broadcasters would pay less, which in-turn would make the content cheaper to the end user as a result.

1.2 Motivation

From working with QUIC multicast on a previous semester, the amount of traction, and good feedback from the industry have been very motivating to keep exploring, and improve this concept. In the hope of contributing to a potential future solution for this QUIC protocol with both unicast and multicast support. Since QUIC is using udp transport which is non reliable, and therefore susceptible to packet loss, an obvious addition to the project is use Random Linear Network Coding (RLNC), to do forward error connection.

1.3 Problem Definition

Therefore, the problem definition for this report is as follows:

How to implement a reliable multicast solution for content delivery using QUIC on any IP supported network?

Chapter 2

Preliminary

This section contains the summary of the state-of-the-art as well as the most important previous research done in the area of audio-video content delivery in *Section 2.2*. *Section 2.1 Random Linear Network Coding/rely* introduces the concept RLNC and the Rely implementation of it. *Section 2.3 Multicast and Unicast delivery* focuses on the concept of content delivery via IP Multicast and IP Unicast, describing today's challenges with content delivery, and presenting state-of-the-art solutions that enhance unicast-based delivery with scalable multicast. *Section 2.4 IP Multicast with QUIC transport* discusses current experiments and latest attempts to combine multicast-based content delivery with the QUIC protocol. Finally, *Section 2.5 Security in Multicast QUIC* outlines security aspects that need to be addressed when discussing deployments of multicast-based architectures.

2.1 Random Linear Network Coding/rely

Random linear network coding is a way of encoding data for transmission, that compared to standard data transfer can inherently handle lost packets. In particular, a standard stream of video is split into many packets, and if one packet is lost, it requires re-transmission, which causes the stream to slow down. RLNC migitates this problem by encoding the original packets by making their random linear combinations and this way creating a stream of encoded packets. Each encoded packet is an equally valid representation of the original packets, and as soon as receiver receives a required number (of any) encoded packets, it can decode the original stream. Overhead of RLNC is close to 0, i.e., if there are N original packets, N encoded packets are enough for decoding with high probability. In effect, loss of an encoded packet is compensated by reception of any other subsequently encoded packet made from the same set of original packets. This is exploited in the way that the transmitter generates more encoded packets then needed, streaming them and combating the losses of individual packets. The benefits of such delivery method are even more pronounced in multicast and broadcast, as the need for individual retransmission and related signalling (i.e., user-specific repair mechanisms) is removed. The price to pay is the complexity of the encoding and decoding algorithms.

An example of RLNC encoding can be seen in figure 2.1. In this example 5 IP packets are encoded, and the data is spread out across multiple packets. RLNC ensures with high probability that if one packet is lost, the data can be recovered using the four other packets.

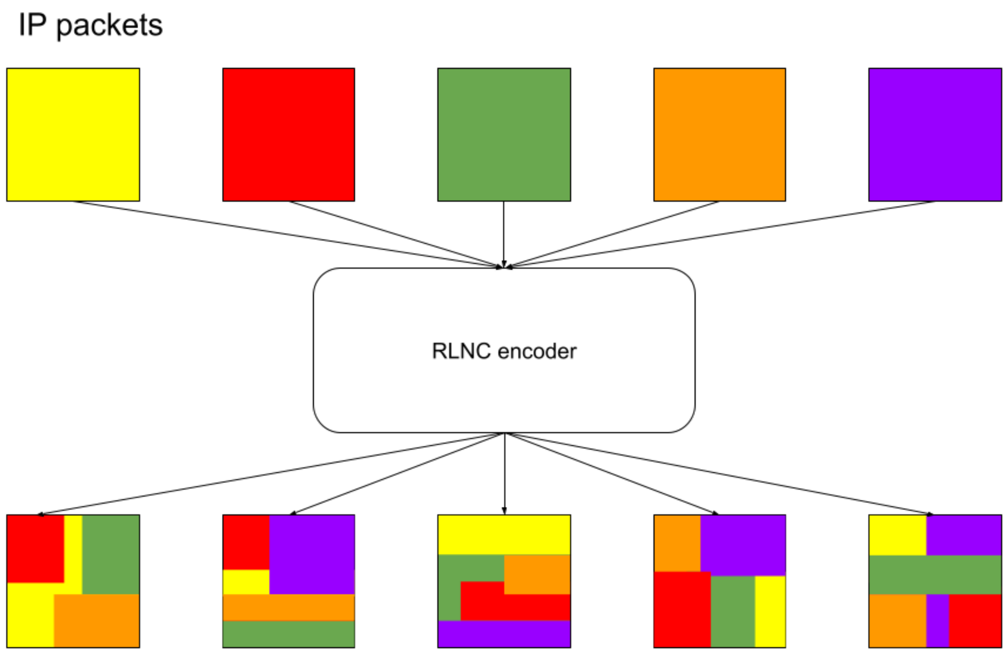


Figure 2.1: RLNC example of IP packets being encoded, created using google draw [11]

Each encoded packet can be seen as a linear equation, where the random scalar coding coefficient can be grouped and viewed as a coding vector. So in the figure 2.1 the colours of the output, is representing that data from each packet is used together to create new data, so the colours should be viewed as mixed together. Similarly, the overall encoding process where multiple coded symbols are generated can be viewed as a system of linear equations with randomly generated coefficients [23]. In practise the RLNC encoding is done by firstly splitting the data for each packet into smaller symbols of equal size. A matrix of all the packets that are being encoded is created, where each row is all the symbols from one packet, this is illustrated in equation 2.1

$$\begin{bmatrix} S_{11} & S_{12} & ... & S_{1k} \\ S_{21} & S_{22} & ... & S_{2k} \\ ... & ... & ... & ... \\ S_{g1} & S_{g2} & ... & S_{gk} \end{bmatrix} \tag{2.1}$$

Now a matrix where each row consist of g coefficients is generated, this is called the coefficient matrix (C). The coefficient vector(c) for each row is derived from a finite field. In short a finite field is also know as a Galois Field (GF), it contains a finite number of elements that follow special rules based on the arithmetic operations required. This guarantees that the result of the arithmetic calculation is always an element that exist in the field [25]. With the coefficient vector found a coded symbol can be constructed by multiplying the coefficient matrix with the symbol matrix. The matrix of the coefficient that results from this can be seen in equation 2.2

$$C_i \cdot S = \begin{bmatrix} C_{i,0} \cdot (S_{11} & S_{12} & ... & S_{1k}) \\ C_{i,0} \cdot (S_{21} & S_{22} & ... & S_{2k}) \\ ... & ... & ... & ... \\ C_{i,0} \cdot (S_{g1} & S_{g2} & ... & S_{gk}) \end{bmatrix} = CS_i \tag{2.2}$$

In this equation each row represents a coded packet, but one more thing is needed to ensure decoding if a packet is lost. In the case of a packet lost, another trick is needed to recover the lost packet, that is done by taking n number of coded packets, and combine them with a XOR operation. The combined packets are transmitted with the coded packets, and can be used recreate lost packets.

A product that provides ready-made and optimized implementation of RLNC for media delivery is Rely from Steinwurf APS [24], which is especially suitable for low latency use cases. The rely encoding is based on a sliding window ECC/FEC algorithm, an illustration of this can be seen in figure 2.2

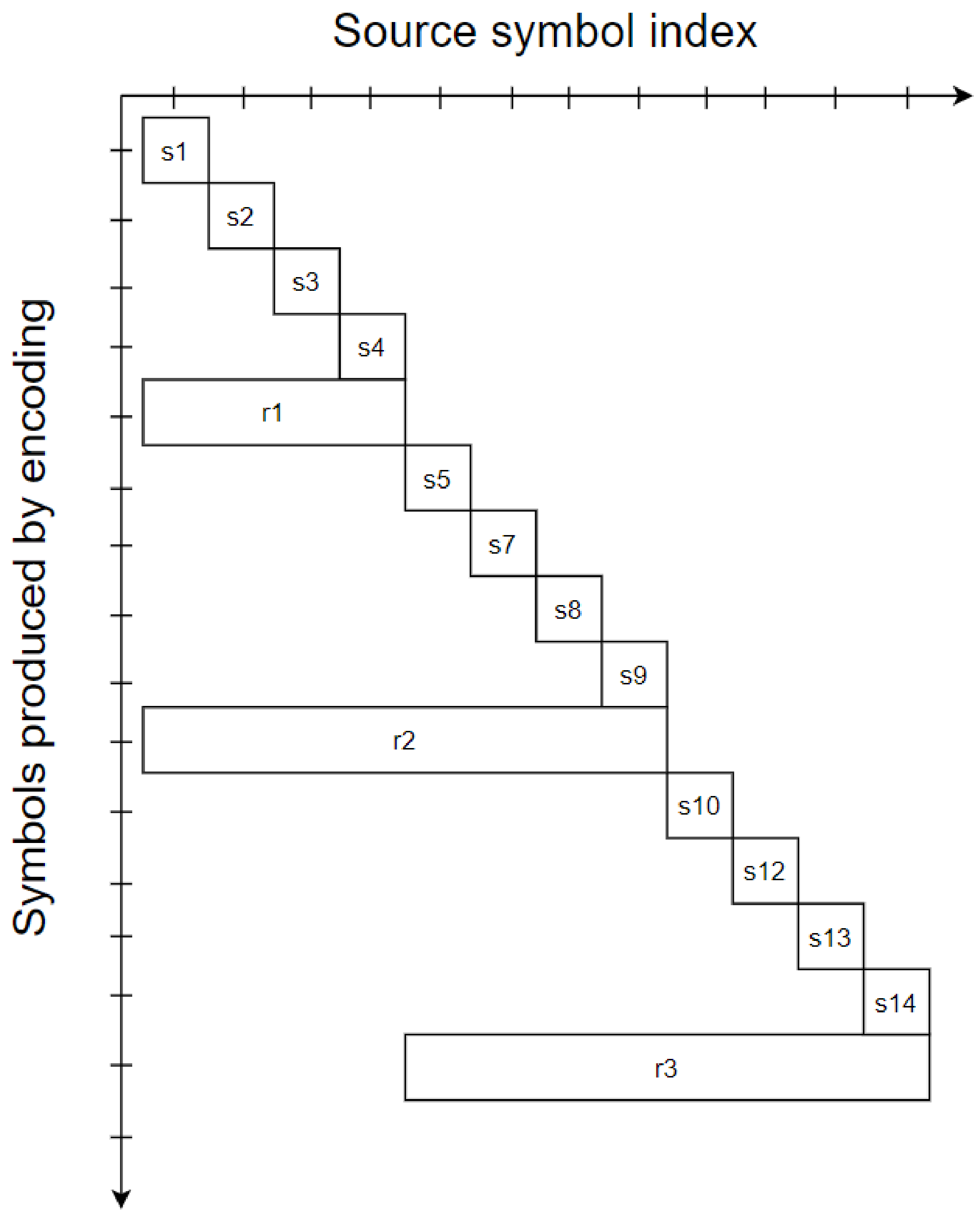


Figure 2.2: Rely encoding window explained illustrated from [24]

In the figure 2.2 the sX is the network packets being added to the encoder, the rX is the repair symbol, where the width of the repair packet shows how many symbols are covered by that repair packet. In this example there is a repair packet every 4 symbols which is equal to a repair rate of 1 / 5 = 20%, this would mean that it could handle up to 20% packet loss during transmission. The repair interval is the main configuration to change, along with repair target. The repair interval is how often a repair packet is generated, while the repair target is the amount of repair packets generated within the interval. So if the previous example had a repair target of 2 it would handle up to 2 / 5 = 40% loss. The bigger the repair interval the more latency is added to the transmission.

2.2 Audio-Video Content Distribution

The following section is based on the same research as the previous project [2]

IP based audio and video distribution is primarily done through either RTP over UDP or HTTP Adaptive Streaming [20]. In both cases the protocols have different advantages and disadvantages when it comes to using them for adaptive streaming. RTP is an application layer protocol built on top of UDP. It is widely used around the world in VoIP and video especially with application for conferencing systems. Usually, it is complemented with session and signalling control protocol like SIP, WebRTC or H.323. It is suitable for applications requiring strict latency and timeliness, while sacrificing the reliability of transmission [20]. Historically real-time audio-video applications choose faster delivery over reliability. As the stability of delivery is not as important as long as the packets arrive in time. In RTP, each packet can be received out-of-order as long as both sender and receiver are in sync with regards to what is contained in the payload of a packet. RTP receivers therefore must be robust to packet loss and need to be able to decode partial streams. This makes RTP implementations for linear content streaming complex, but allows it to operate in sub-optimal network conditions, and are one of the main disadvantaged compared to HTTP adaptive streaming [20].

Streaming applications that allow for a more relaxed latency bounds will generally make use of HTTP adaptive streaming over TCP i.e. MPEG DASH due to the ease of deployment on commodity CDNs. HTTP-based transports are unfortunately not optimised for media delivery [20]. In HTTP adaptive streaming, server encodes chunks of media data at different bit rates, and exposes HTTP endpoints allowing clients to fetch those according to shared manifest that indexes the available content.

There are currently three most commonly used standards for low latency HTTP adaptive streaming: DASH-LL, LHLS, LL-HLS. They all focus on HTTP delivery, and require content to be chunk encoded, delivering end-to-end latency of 2-10+ seconds [15]. Furthermore, DASH-LL and LHLS both support chunk transfer encoding, allowing the the receiver to start decoding chunks inside segments before they have been fully sent out from the origin to the CDN. In particular, in DASH-LL flow, player requests a MPD (Media Presentation Description) file, which describes available content representations as segme nts of pre-defined length. These segments are contained within a CMAF(Common Media Application Format) containers that split segments into chunks. Chunks of content are smaller than full segments, and allow the player to request smaller portions of content more often when the connection quality is low, improving the perceived latency at the player [15]. The general familiarity of developers with HTTP semantics is desirable, however since TCP transport itself is reliable, ordered and congestion controlled, clients need to employ large receiver buffers to hide the variation in download times to allow stall-free play-out, making HTTP adaptive streaming a poor choice for low-latency play-out of live content.

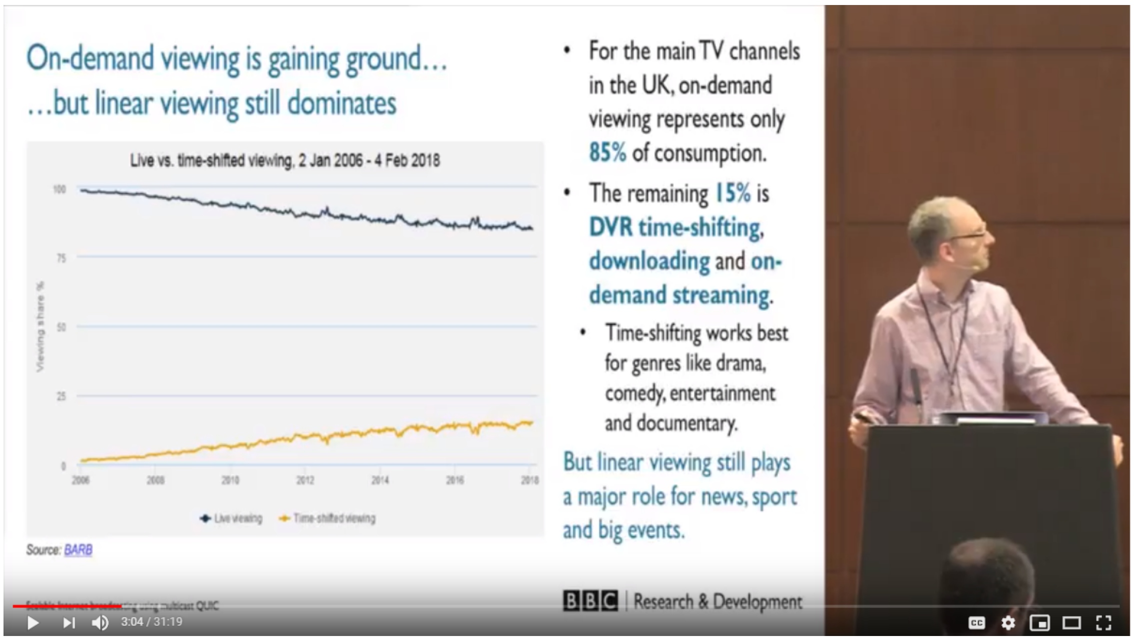
The familiarity of HTTP semantics is considered a major advantage of HTTP based application protocols [5] [26] [20] [19]. This is reasonable, as HTTP/2 is used by approximately 44.2% of all of the websites [28]. HTTP/2 multiplexes many logical streams under one physical TCP connection. It was a major improvement from HTTP/1.1, as websites could reduce the amount of open client-server connections. This allows applications to make hundreds of thousands parallel transfers over a single TCP connection, drastically improving web performance [27]. Although the solution fit the past web usage well, the rate of consumption of streaming content changed drastically from the time HTTP/2 was introduced. For streaming applications, TCP protocol introduces a great limitation, the head-of-line problem [20]. After the TCP connection has been established, application can transmit TCP frames over the IP protocol. TCP delivers reliability through re-transmissions of frames either damaged or lost in traffic. TCP frames are always delivered in order they were sent, regardless of how many streams are multiplexed within the connection. That means that loosing even a single frame blocks the entire connection until it's re-transmitted and acknowledged by the receiver. Addressing head-of-line problem within the TCP protocol itself is extremely difficult if not downright impossible [27].

A major challenge in addressing problems related to TCP protocol is labeled by professionals as protocol ossification [27]. Nowadays, Internet traffic runs through many nodes like routers, proxies, gateways, switches and such. Before the traffic ends up at the destination, and sometimes hops through tens if not hundreds of such devices. Software on those nodes have historically shown to lack behind the innovation in transport protocols. Many of them were deployed a long time ago, with security standards fitting the old days [27]. Often they simply reject the traffic that is unknown to them, or falsely categorizing it as malicious. It could be TCP traffic with unrecognised TCP headers, or packets associated with a newly developed, innovative transport protocol.

2.3 Multicast and Unicast delivery

The following section is based on the same research as the previous project [2]

To provide video streaming content on a large scale is very expensive, the costs of using public CDNs are raising exponentially with the amount of tuned in viewers [5]. The more popular the content is, the more expensive it is to distribute. Yet, the potential demand for linear content still dominates the market (see figure 2.3). Since the potential audience for linear content continues to grow, there is a strong market push from the side of broadcasters to bring distribution costs down. Due to how encoding can save a lot of data, live streaming in high quality is very data heavy, especially if low latency is an requirement.



On-demand viewing is gaining ground…
…but linear viewing still dominates

- For the main TV channels in the UK, on-demand viewing represents only 85% of consumption.
- The remaining 15% is DVR time-shifting, downloading and on-demand streaming.
  - Time-shifting works best for genres like drama, comedy, entertainment and documentary.

But linear viewing still plays a major role for news, sport and big events.

Source: BARB

Figure 2.3: Showing continuous demand for linear content from a BBC presentation [5]

Distribution costs further raise from ever-raising demand for High Definition technologies, like higher spatial resolutions or 4K, better color fidelity with HDR, better motion depiction and frame rates. There is also a demand for services delivering new content experiences like AR (Augmented Reality), VR (Virtual Reality) or 360 degrees video [5]. Many broadcasters are still operating within a limited spectrum, which is being increasingly dominated by cellular technologies. There is a huge need for delivering broadcasting services through the Public Internet. Video has become a dominant class of traffic on the public networks [10]. Market has widely adopted unicast methods of delivery due to high reuse of existing network technologies to match the demand at scale [10]. Content Delivery Networks have built on top of unicast technologies for years, and through the innovation of streaming through HTTP, they can gracefully degrade transmissions using Dynamic Bit Rate Adaptation. The approaches of the past do not meet the requirements of the present. Today, the same content is often consumed by masses at scale, creating multitudes of redundant connections between the network edges and the players requesting the same payload [10]. A single video streaming session for each user puts Quality of Experience at risk [10]. This can be solved by point-to-multipoint approaches using IP Multicast for scalable and efficient linear media distribution [10]. Instead of pushing stream elements to the players individually, using IP Multicast, it is possible to push stream elements into a single Multicast group address, allowing the players to subscribe to the Multicast stream and receive the same content via packet duplication. Reduction of redundant server connections and preemptive asset delivery closer to the user can greatly reduce public network congestion, at the same time bringing cost of media distribution down by pushing content closer to the user ahead of time [5]. Additionally, streaming an entire channel using a single session provides great economy of bandwidth [10]. DVB-I is a state-of-the-art standard for Digital Video Broadcasting over the Internet [10]. It describes potential architectural options for video distribution via IP Multicast, supported by Unicast Repair Mechanism, which is capable of filling the stream elements lost in transit via traditional Unicast channels on demand.

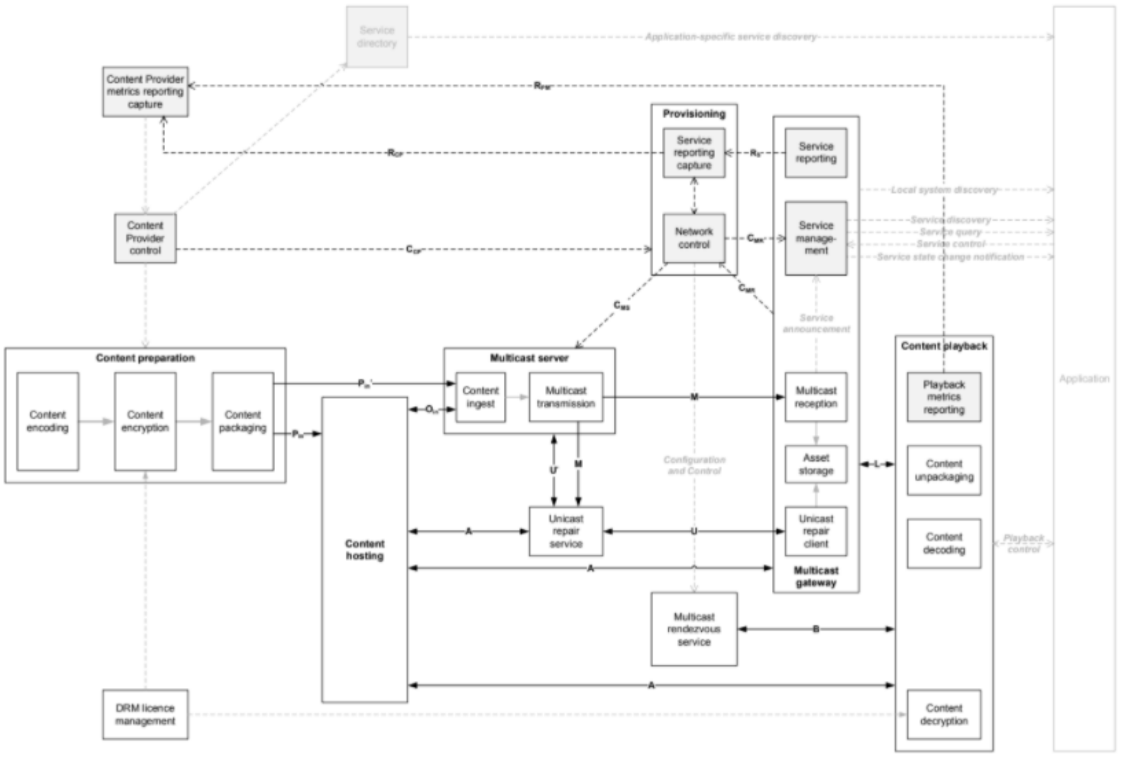


Figure 2.4: DVB-I - Adaptive Media Streaming over IP Multicast, Reference Architecture [10]

Content Delivery Networks prepare content for distribution by encoding it into chunks send interchangeably via adaptive bit-rate functionality, encrypting using

DRM encryption keys and packing into desired media distribution format. The format differs depending on the HTTP streaming protocol used. Prepared content chunks are made available for unicast delivery to the playback services. DVB-I introduces a number of critical components designed for multicast delivery, Multicast Server, Multicast Gateway and Unicast Repair Service. These services allow playback services to receive content transported via multicast.

- **Multicast Server** serialises and transmits content chunks as streams inside IP packets payload towards the Multicast Gateways via multicast. Can Push/Pull using HTTP, or push via RTP.
- **Multicast Gateway** provides packaged content segments to the playback services using built in-memory cache storing and positioning content chunks and advertising assets. Can be a forward proxy or a local origin including reverse proxy. It could be installed in customer premises inside home gateway devices or IP-connected set-top boxes. It could also be located in an upstream network node as an alternative to the customer premises. Unicast fill operations are performed until cache is established in Asset Storage for a given linear service. Cache takes some time to establish. Used for pre-positioning of media content assets (popular assets of advertising material pre-positioned in full or partially in advance due to large population of users). Also used for temporary caching of linear media content segments.
- **Unicast Repair Service** listens to multicast content transmissions, and locally caches a copy of packet stream. When the repair request arrives from the Unicast Repair Client, then the pre-cached chunk is retransmitted. If the cache is missed, the packet repair request is passed to the Multicast Server. Unicast Repair Service can also request the repair via HTTP(S) from content delivery network. If the Unicast Repair Service receives many repair requests of the same packet from many Multicast Gateways, it can then start transmitting those via the Multicast channel.

2.4 IP Multicast with QUIC transport

The following section is based on the same research as the previous project [2]

QUIC protocol can be used in conjunction with IP Multicast architecture to address mass audiences reliably using managed and unmanaged networks [5]. The means to bulk transfer resources over Multicast IP using HTTP semantics presents an opportunity to more efficiently deliver services at scale, while leveraging the wealth of existing HTTP-related standards, tools and applications [14]. Using HTTP and QUIC reduces client complexity by adopting common network protocols across unicast and multicast delivery modes [5]. Both unicast and multicast delivery modes can be operated using common media packaging (ISO Base Media File Format, fragmented MP4) to reduce operational costs [5]. MPEG-DASH with multiple representations being transmitted through multiple channels lets the architecture react to dynamic network conditions using dynamic adaptation techniques [5].

In a proof-of-concept of Multicast QUIC created by R&D Department of BBC [5], visible in the Figure 2.5, multicast delivery was supported by unicast repair using HTTP, while both delivery modes were performed through QUIC. Service discovery was done using HTTP/2 Alternative Services header pointing to the available HTTP/3 server serving QUIC connections. The header also included certain parameters required for connection upgrade.

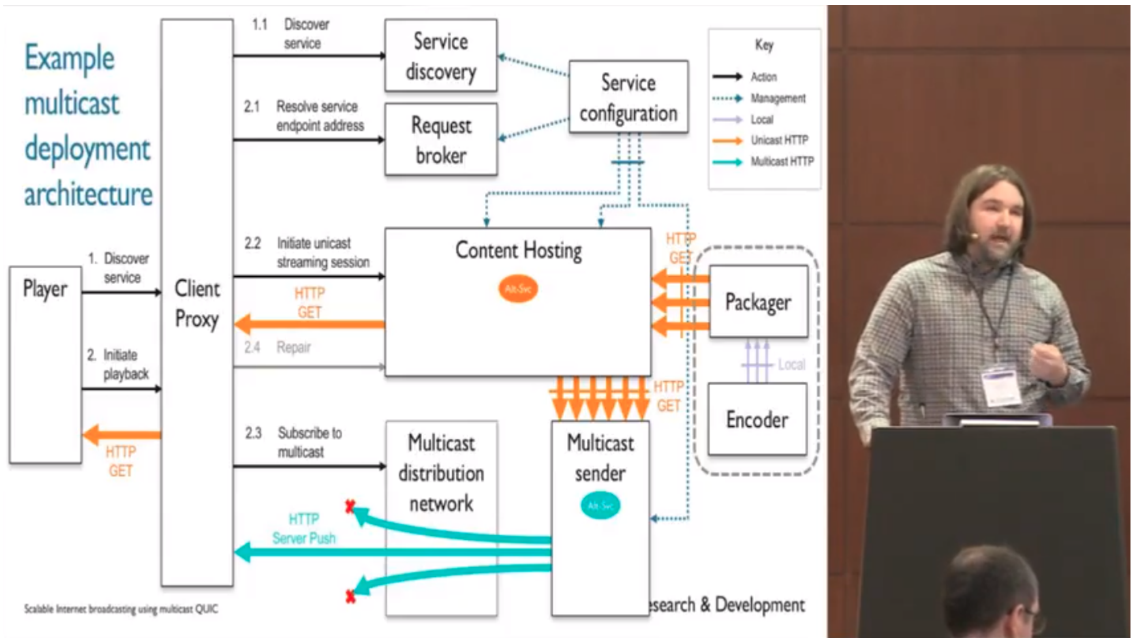


Figure 2.5: BBC Content Distribution Architecture using Multicast QUIC [5]

On initiation of the unicast streaming session, traditional HTTP Adaptive Streaming server includes and Alt-Svc header, through which it communicates the location of the alternative HTTP/3 server that the client can use to listen to the IP multicast stream via QUIC [14]. This is referred to as a QUIC hand-off. As soon as the hand-off is done, the origin server starts provisioning elements of MPEG-DASH stream session in IP Unicast, while in the background, the multicast session is being verified and established. Once the client determines that it can use the multicast service, it will subscribe to the *Multiast Distribution Network*. The content will then flow in multicast from the *Multicast sender* to the client using HTTP/3 Server Push, an interaction mode introduced in HTTP/2 which permits a server to pre-emptively push a request-response exchange to a client in anticipation of the client making the indicated request [3]. Since Server Push frames cannot be sent from the server unless client explicitly requests for it [3], the QUIC hand-off acts as both service discovery and Server Push initiation mechanism. Unicast is used to retransmit lost packets that might occur in the multicast stream.

2.5 Security in Multicast QUIC

The difficulty in securing a multicast connection is in the core of how the multicast packets are distributed. Because the multicast server is sending a stream of data to a group of receivers, the packets are replicated by the routers in the network. This is done so the sender can send one packet once, instead of sending multiple of the same information to each recipient. The problem is then encrypting the packet so only recipients with the credentials can read it. Traditionally in unicast connection this is solved by a key exchange in the initial TLS handshake, this is not possible in multicast connection [12]. One of the solutions for security of Multicast Transmission involve creating Security Associations [12]. Whenever a new member joins the multicast group [12], the dedicated shared key is created that can be used by the member to decrypt the transmission. This requires creating and managing a pool of session keys, which is not optimal due to low scalability and reliability of the solution, especially if there is many leave/joins calls in the multicast group. The QUIC hand-off session discovery mechanism can be used by the receivers to obtain a session decryption key. An illustration of this can be seen in Figure 2.6.

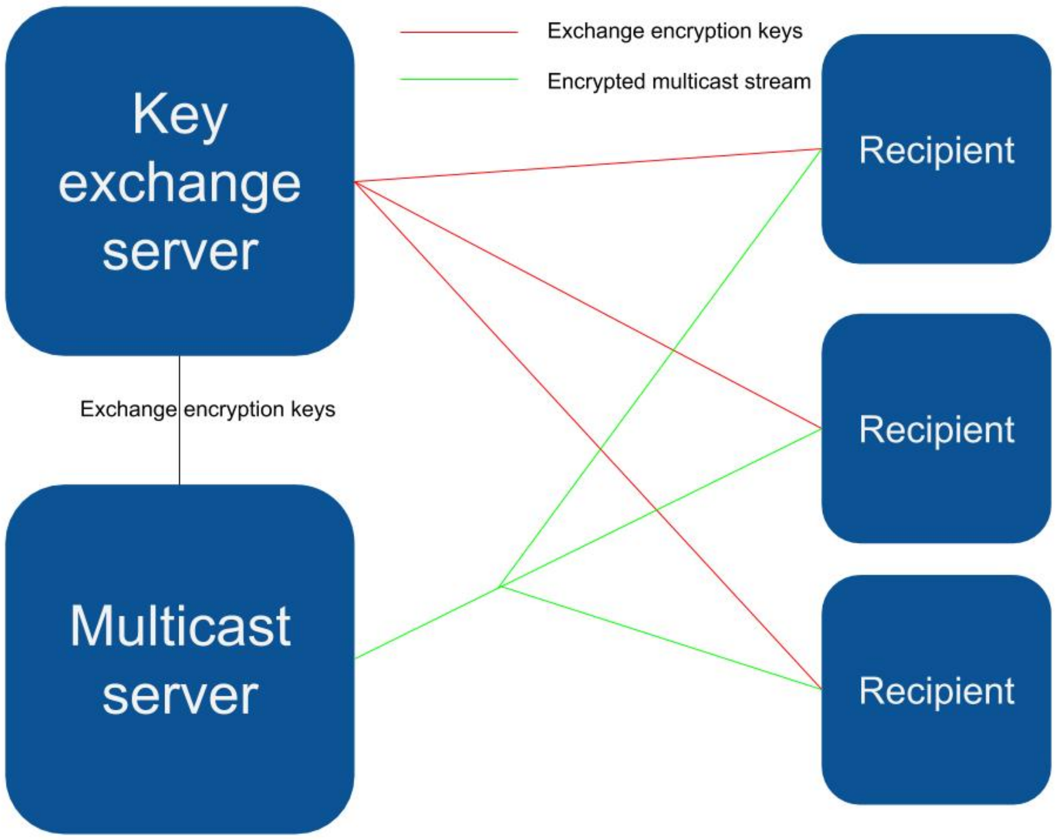
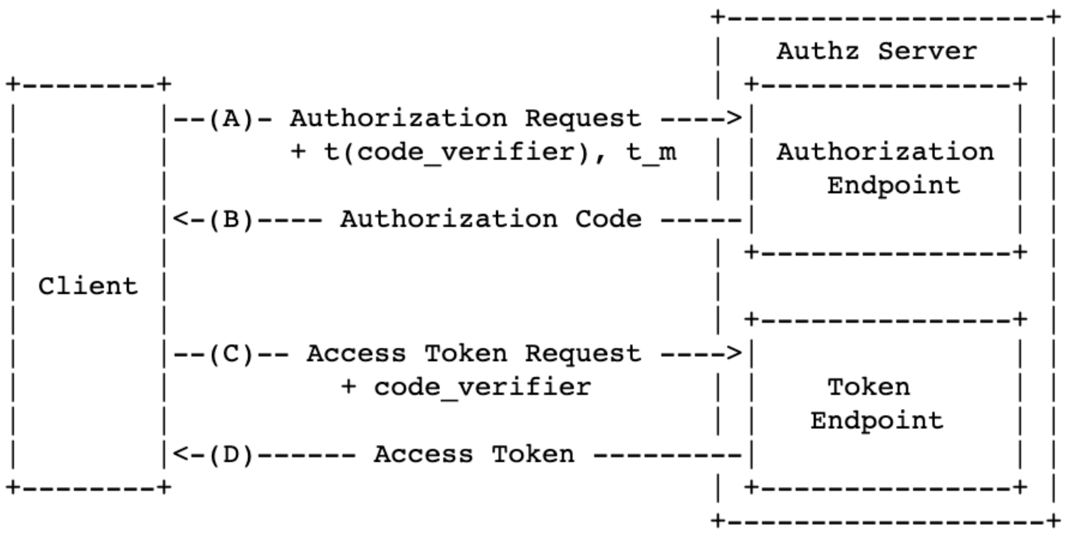


Figure 2.6: Key exchange proposal - created using google drawings [11]

Using a shared key for session encryption is not a strong protection [14]. Therefore, Multicast QUIC session advertisements should be conveyed over a secure transport that guarantees authenticity and integrity in order to mitigate attacks related to a malicious service advertisement, for example a "man in the middle" directing endpoints to a service that may lead to other attacks or exploits [14]. This also prevents receiver spoofing attacks as the QUIC hand-off happens through the authenticated and secure channel using TLS 1.3. Sender Multicast traffic is at risk of being spoofed if the malicious actor obtains session information and the shared key, therefore the applications should employ content authenticity mechanisms [14]. Leaking of the shared key makes the application vulnerable to further replay attacks. Deletion of messages is partially mitigated by the unicast repair fallback mechanism, although it might impact performance [14]. There also exists a risk of stampeding herd of unicast repair requests opportunistically occuring on the side of service provider due to some unexpected network events. Finally, malicious monitoring of the unicast repair mechanism on the receiver side can be exploited, leading to the leakage of user behaviour data [14]. An attacker could therefore gain insight into any of the multicast session participant by monitoring the TCP port of it's respective unicast repair counterpart. However, knowledge that a user (or group of users) has participated in a session is sensitive and may be obtained by correlation between with observable multicast and unicast traffic. Applications concerned with this risk of data leakage should completely disable support for unicast repair, at a cost of reduced service quality [14]. By doing common encryption channels could be distributed with a encryption that requires some special hardware key to decrypt, like there is in various dvb standard with smart cards [29]. To enable viewing from locations without internet access.

According to DVB-I, Multicast Gateway authentication currently does not exists [10]. However, a sufficient Gateway authentication gateway can be achieved by combining OAuth 2.0 [13] flows with PKCE (Proof Key for Code Exchange) [18] mechanisms. The OAuth 2.0 requires an Authorization Server that is capable to advertise authorization tokens to the client, and exchange those for access tokens. The Proof Key for Code Exchange framework ensures that only the application that requested the authorization gain access to the access token [18]. This is ensured by generating *core-verifier*, a high-entropy cryptographic random String or ASCII characters, between 43-128 characters. This string is further transformed into a Base64 encoded SHA256 hash, called *code-challenge* [18]. The Authorization Server associates the initial authorization request with the client ID, selected code challenge and the challenge method, and verifies that the client returns correct code verifier along with previously issued authorization code. This flow is presented in the Figure 2.7.



```
                                                 +-------------------+
                                                 |   Authz Server    |
+--------+                                       | +---------------+ |
|        |--(A)- Authorization Request ---->|               | |
|        |       + t(code_verifier), t_m     | | Authorization | |
|        |                                   | |   Endpoint    | |
|        |<-(B)---- Authorization Code -----|               | |
|        |                                   | +---------------+ |
| Client |                                   |                   |
|        |                                   | +---------------+ |
|        |--(C)-- Access Token Request ---->|               | |
|        |          + code_verifier          | |    Token      | |
|        |                                   | |   Endpoint    | |
|        |<-(D)------ Access Token ---------|               | |
+--------+                                   | +---------------+ |
                                             +-------------------+
```

Figure 2.7: Oauth 2.0 with PKCE authorization flow [18]

Chapter 3

Analysis

The following section reveals the proposed solution for low latency content distribution using IP Multicast with QUIC protocol. The detailed documentation of the solution is described in the *Section 3.1 Overview*, which explains the specifics of Multicast-based content delivery over QUIC as well as security considerations for such a solution. Finally, the *Section 4 Implementation* presents a first iteration of the solution implemented with Go programming language.

3.1 Overview

The main implementation is creating a new protocol, forked from the existing QUIC-Go [7] library, utilizing all the existing features from unicast connection, such as TLS 1.3, and reliable UDP. The challenge is then to create a multicast stream, that works in unity with the unicast stream, so one stream can be over the traditional internet, and another on a managed network. Where as traditional media distribution over IP, is routed through the internet over a unicast connecion. The public internet works in a best-effort environment, with many nodes owned by many different organisations, where multicast traffic is blocked by default.

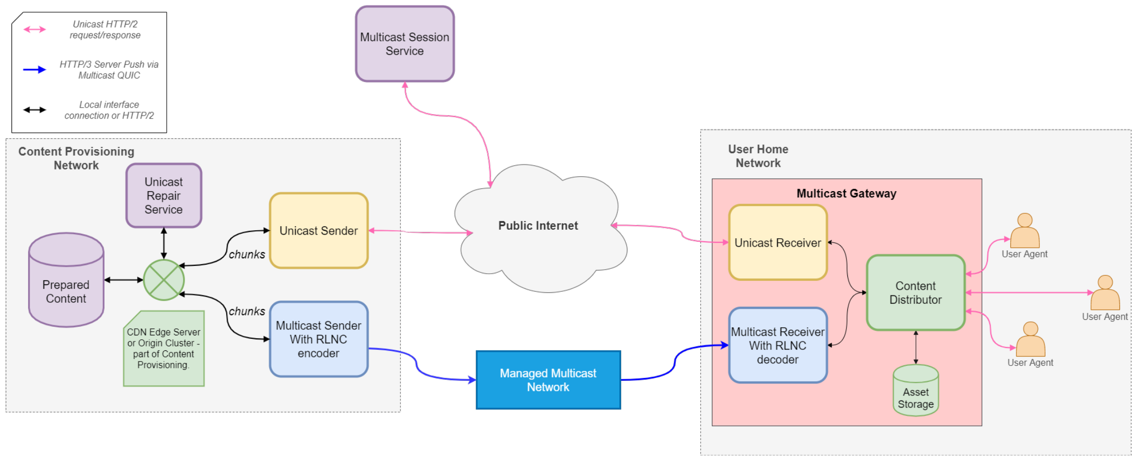

Unicast HTTP/2 request/response
HTTP/3 Server Push via Multicast QUIC
Local interface connection or HTTP/2
Multicast Session Service
Content Provisioning Network
Unicast Repair Service
Prepared Content
Unicast Sender
chunks
chunks
Multicast Sender With RLNC encoder
CDN Edge Server or Origin Cluster - part of Content Provisioning
Public Internet
Managed Multicast Network
User Home Network
Multicast Gateway
Unicast Receiver
Content Distributor
Multicast Receiver With RLNC decoder
Asset Storage
User Agent
User Agent
User Agent

Figure 3.1: Context Diagram showing overview of QUIC Multicast powered content delivery, created with diagram.net [9]

The context diagram in the figure 3.1 shows an overview of the architecture for Multicast QUIC content delivery system. The novelty of the proposed solution is in the transportation method of using QUIC over multicast with RLNC encoding. The Multicast QUIC traffic can be received by multiple gateways that are part of the managed multicast network. The receivers are not the end user, but rather a physical setup boxes deployed on customers premises, or small servers that are managing connection for multiple end clients. Setup boxes act as an intermediary proxies for the HTTP adaptive streaming traffic between *Content Provisioning* and the *User Agent*. The *User Agent* is no longer requested directly through the Internet, but rather through HTTP/2 Unicast endpoints exposed on the *Multicast Gateway*. The Gateway itself forward the request from the user, and starts ingesting content via Unicast HTTP or QUIC and Multicast QUIC. This approach allows the content to be transmitted as close to the user as possible, using the innovative transportation method of IP Multicast through QUIC, while keeping the end-user client complexity low. The primary driver for such deployment is interoperability, allowing many different browser and web applications to consume the content as they would normally. Presented solution allows devices within the home network to consume streaming media content delivered through means of Multicast QUIC, while keeping the convenience of utilizing traditional mechanisms of HTTP/2 based Unicast delivery via TCP. The main benefit of using Multicast QUIC delivery of content from Content Provisioning to the *Multicast Gateway* is that it decreases the number of active Unicast streams. Instead a much more scalable Multicast network deliver the content to possibly hundreds or thousands of *Multicast Gateways* at the same time, while preserving the Unicast HTTP/2 semantics in the last-mile. *User Agents* again need only a minor upgrade to start utilizing Multicast QUIC. The solution is tailored towards deployments of low latency HLS HTTP adaptive streaming [5], however, it can be adjusted to support other protocols like MPEG-DASH [15].

3.2 Use Cases

Some of the use cases for this technology, are all mainly revolving around video streaming, due to the high bandwidth use, and in some cases low latency requirement. But the technology could also be used in a content delivery network for files other than video, such as software updates. Because of the high versatility of QUIC, it can also be deployed in many places, and not just be a part of peoples home setup. The use case of having a super clients in 4g/5g mobile networks, in the actual base station, can enable mobile users, much higher quality and lower latency of a shared video stream. For example a big sports event might be viewed by many people all connected to the same tower, and takes a lot of the towers bandwidth. In the figure 3.2, this example is illustrated, where the use of multicast would only use the bandwidth for one video stream, compared to unicast where it would be multiplied with the amount of devices streaming.

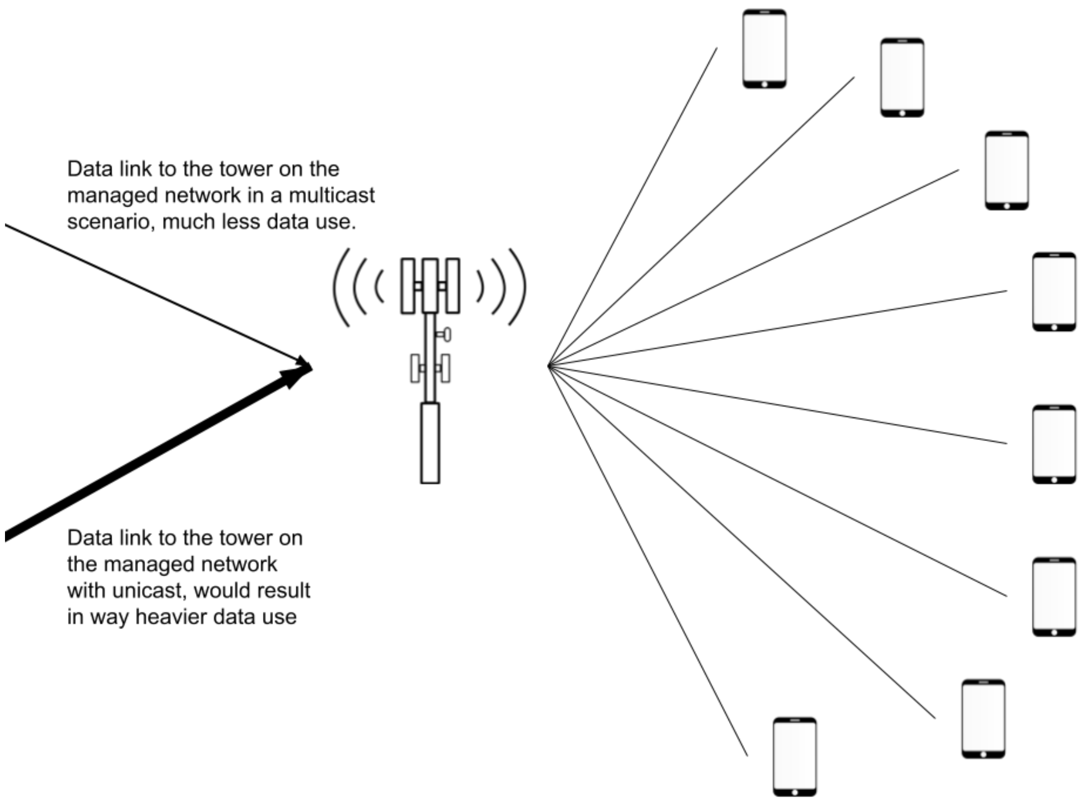

Data link to the tower on the managed network in a multicast scenario, much less data use.
Data link to the tower on the managed network with unicast, would result in way heavier data use

Figure 3.2: QUIC super client in a mobile broadband tower, unicast vs multicast, created using google draw [11]

The use cast in figure 3.2, would mostly be relevant during big live events, where many people are watching the live stream, but could also be used to potentially push phone updates. The biggest benefit would be that collectively in a big network, the latency compared to watching on phone, and traditional broadcast TV should be much closer. Where as it is today, a popular unicast stream of live sports events can add a lot of latency compared to broadcast [8].
The trend of higher and higher resolutions in TV, when only very few live events are streamed in 4k [17], as the bandwidth required is very expensive. This is therefore a very good use case for having a multicast network, where high resolution content can be distributed cheap and efficiently, especially in the future when 8k, VR or 360° live streaming is more common. This solution will scale very well in situation where many people are connected to the same internal network, for instance on a ship, or in a plane, the passengers and crew could watch sports events live in

high quality. Another use case could be deployment in rural areas, where getting video stream in high quality might be impossible if the bandwidth is too low. The low use of bandwidth for the unicast connection is still needed, but the data use is minuscule compared to the bandwidth for video. This could lead to new use cases of consumption of media, as the multicast gateway, can cache content for watching later. Though the unicast connection could be removed completely if a smart card or something similar can decrypt the stream, this would obviously not work with the features from QUIC.

3.3 Requirements Specification

This section contains the requirements for the architecture of Multicast QUIC protocol, that can reliably transfer data over any IP supported network. The architecture is tuned for delivery of audio-video using any low latency protocol for adaptive bit-rate streaming. This section defines base reference components of the architecture and their functions within it. The reference architecture describes architectural components and relationships between them. It presents out the primary elements required for engineering of a Multicast QUIC content delivery system, and explains channels through which data flows between the components. The architecture is show in the figure 3.3.

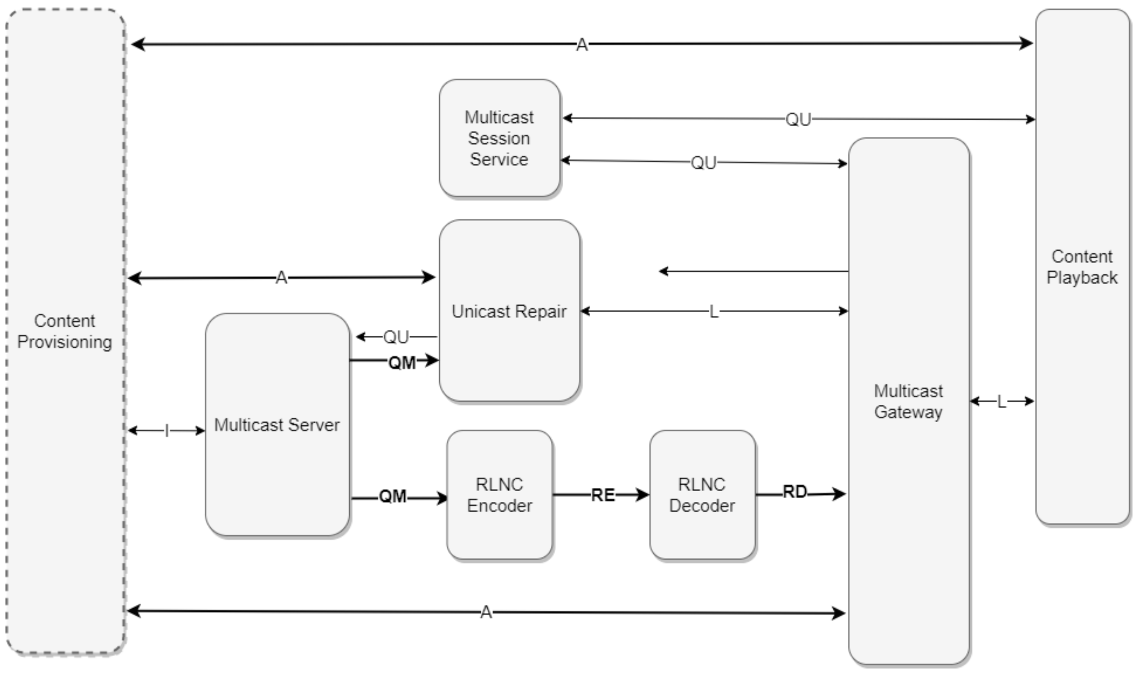


Figure 3.3: Reference Architecture of QUIC Multicast

Architectural Components

This section presents basic architectural components of Multicast QUIC architecture.

1. **Content Provisioning**: a network that exposes prepared content. Prepared content is chunk-encoded, encrypted using DRM license keys and packaged into ISO Base Media File (Fragmented MP4) audio/video files.

2. **Multicast Server**: an HTTP/3 web server deployed within the private network that ingests content from *Content Provisioning* and exposes QUIC endpoints allowing clients to subscribe to Multicast streams.

3. **Multicast Session Service**: service deployed on the public or private network that manages QUIC Multicast sessions between *Multicast Server* and many *Multicast Gateways*. It handles QUIC hand-off process by keeping a record of active and historic sessions, and verifying the authenticity of *Multicast Gateways* wishing to receive Multicast QUIC streams.

4. **Unicast Repair**: have unicast repair on packet level as part of the protocol, that can deliver packets that were damaged or missing in transit because of the instability of the Multicast channel due to i.e. packet loss. Unicast Repair differs from the unicast repair service from BBC concept as mentioned in section 2.4.

5. **Multicast Gateway**: device capable of receiving LL-HLS via Multicast QUIC, supported by the *Unicast Repair* functionality. It holds an *Asset Storage* that contains advertising assets as well as preemptively loaded streaming chunks. Finally, it exposes HTTP/2 Unicast endpoints that serve chunks to *Asset Storage.*

6. *Content Playback*: software capable of receiving, unpackaging, decoding and decrypting LL-HLS audio/video stream via HTTP/2 Unicast.

Data Plane Interactions

This section presents data plane interactions and interfaces between components of Multicast QUIC architecture.

1. **A**: HTTP(S) acquisition of chunks. Used by *Content Playback* to request content via HTTP/2 Unicast from *Content Provisioning.* Used by *Multicast Gateway* to receive content via HTTP/3 Unicast when the Multicast session is not yet established. Used by *Unicast Repair* to repair discarded chunks via HTTP/3 Unicast during the Multicast transmission.

2. **QM**: HTTP transmission of chunks via HTTP/3 Multicast. Used by *Multicast Gateway* to receive audio/video chunks from *Content Provisioning* through the Multicast interface. Interface between *Multicast Gateway* and *Content Provisioning* must be established through means of QUIC hand-off through *Multicast Session Service.*

3. **RE**: RLNC encoder, that encodes the multicast stream before being sent out.

4. **RD**: RLNC decoder, that decodes the multicast stream before relaying the packets to the multicast gateway.

5. **QU**: HTTP/3 Unicast transmission between ends of classical HTTP communication. Used by *Multicast Session Service* to receive session parameters for Multicast QUIC session from *Content Playback* that have received such through means of HTTP Alternative Service QUIC hand-off. Additionally, the *Multicast Session Service* uses it to verify the authenticity and status of *Multicast Gateways*, and to delegate the Multicast QUIC streaming session to the Available Gateway.

6. **L**: a local API or a HTTP(S) interaction that transports the LL-HLS stream. Used by *Multicast Gateway* to deliver chunks to *Content Playback*.

7. **I**: Content ingest that transports prepared content from *Content Provisioning* to the *Multicast Server*, typically a pull interface.

Functions

This section enumerates individual functions of the architectural components.

Content Provisioning

This service might be a simple web-server, a part of the origin cluster, a distributed Content Delivery Network where the classical load-balancing and request distribution techniques apply.

1. **Content Encoding**: transforms source media streams into encoded media for bit-rate reduction. Output could be in MPEG-2 Transport Stream or any other proprietary intermediate format in cleartext.

2. **Content Encryption**: encrypts the encoded media streams using DRM license keys, returning a cyphertext stream.

3. **Content Packaging**: packages the media segments into ISO Base Media File Format (Fragmented MP4), returning a sequence of packaged media segments with representation switching points that are aligned across the different representation of the same source media.

4. **Content Hosting**: Exposes media content for Unicast transmission to the *Multicast Server* via the interface I, to the *Multicast Gateway* for missed cache

and to the *Unicast Repair Service* through the interfaces A, or to the *Content Playback* instances that are not connecting through the *Multicast Receiver*.

Multicast Server

This server is an HTTP/3 Server, which location is advertised to clients through the QUIC hand-off service discovery via the Alt-Svc header.

1. **Content Ingest**: ingests content from the *Content Provisioning* via the pull interface I. The exact specification of the pulling method is outside the scope of this report.

2. **Content Transmission**: transports ingested content stream packaged into a QUIC stream via the interface QM to the subscribed *Multicast Gateways*.

Unicast Repair

This can be an independently deployed service for handling many Multicast Gateways or as part of the Multicast Gateway.

1. **Payload Repair**: offers a Unicast Repair functionality to the *Multicast Gateway* via the interface *L*. The repair service listens to the Multicast transmission via the interface *QM* and locally caches copies of streamed packets to be used when the Unicast repair request arrives. If the cache misses, payload request is requested from the *Content Provisioning* via the interface A. Alternatively, the repair request can be passed to the *Multicast Service* for simultaneous delivery to many subscribed *Multicast Gateways* using the interface *QM*, which is especially effective if there are many duplicate requests for the same chunk.

Multicast Gateway

May be realised as a forward proxy or as a local origin (reverse proxy). It could be installed on customer premises inside an IP-connected setup box or built into a home gateway device.

1. **Service Management**: collects Multicast session information.

2. **Unicast fill Reception**: Delivers media segments to *Content Playback* in the Unicast mode until the Multicast session is fully established.

3. **Multicast Reception**: ingests media content segments through the Multicast QUIC using the interface QM. The QM interface is created only through an explicit authorization from the *Multicast Session Service.*

4. **Asset Storage Control**: controls the temporary storage of assets installed inside the *Multicast Gateway.* Caches media segments transmitted either via Multicast QUIC through the QM interface or the Unicast QUIC through the A interface. Stores pre-positioned media assets made available to the Gateway pro-actively due to high popularity of certain content as well as advertising material.

5. **HTTP/2 re-transmission**: deliver the cached assets over HTTP/2 to the end user playback device, such as a browser or TV. This essentially translates the Multicast QUIC to Unicast HTTP/2, for compatibility for all devices.

Content Playback

1. **Content Unpacker**: extracts elementary stream data from the retrieved transport object.

2. **Content Decryption**: decrypts the content using the DRM keys.

3. **Content Decoding**: parses and interprets the elementary stream data into renderable audio and video.

Chapter 4

Implementation

In this section the implementation is explained in detail, with highlights to key features. The implementation is based on the golang project quic-go [**quic-go**], with many modifications required to make it work with multicast.

Protocol

To make the requirements work from section 3.3, many parts the existing QUIC protocol needs to be completely changed. The only parts that are the same is how the tls handshake is established, which when done still is a bit different because it sends the key required to decrypt the multicast stream. Enabling multicast is not that difficult if the multicast communication is bidirectional, the difficult part is to make the multicast work unidirectional in sync with a bidirectional unicast connection. Some key areas to make this work is keeping track of the quic connection stream, when ever a new connection is created, it also creates new streams. Normally the server would create a new stream for each client, this stream can change or replaced for any number of reasons. But that does not work with multicast, as the client will not trust any random stream, so before any clients connect to the server, a fixed multicast stream is created, ready to send data. This does in turn change more or less every component used to build the packets, as each packet also needs to be acknowledged, and be transmitted if lost.
QUIC works with sessions, so whenever a connection is created a session is created to handle handshake, retransmission queue, send queue, packet packer, connection flow and more on both the client and server side. These just a few components part of the session, but they are the more crucial to enabling the multicast support.

Retransmission queue: Each packet generated has to be acknowledged, or else they get added to the retransmission queue, but retransmitting a packet sent from a different stream.

Send queue: The packet packer sends packet to the queue, the send queue will depending on the packet and state of the server, choose if the packet is sent via unicast or multicast.

Packet packer: packs the packets with data, assigns a packet number, also tags each packet if it is for multicast transmission.

Connection flow: Consists of multiple components that all contribute to figuring out bandwidth, latency(RTT) and packet loss.

Another part of QUIC is round trip time, and how that effects the bandwidth of the transmission, this is normally calculated each time the server receives the acknowledges packets, but since the multicast stream and unicast stream latency can be very different, it changes many parts of this calculation. The solution leads to having a fixed target bandwidth for the multicast stream, as it being impossible to target one that would fit for all the receivers. The fixed bandwidth will off course vary from each network it is deployed in, but can be a benefit in a managed network, where traditionally each TV channel would have a fixed bit rate as well, and might be better solution overall.
For the clients to handle these the multicast handoff, with key sharing some custom headers are also required in the response writer. The client starts by requesting the manifest file, and after the handshake, the multicast connection is opened, and the data is listened to. But only saved when a new file is sent out, which in turn means that the client also request the first few video segments, in order not to have any hick-ups in video playback. A class diagram with many of the components can be seen in figure 4.1, the full class diagram can be seen here[1]

[1]Full svg class diagram: `http://jonessoftwaresolutions.com/class_diagram.svg`

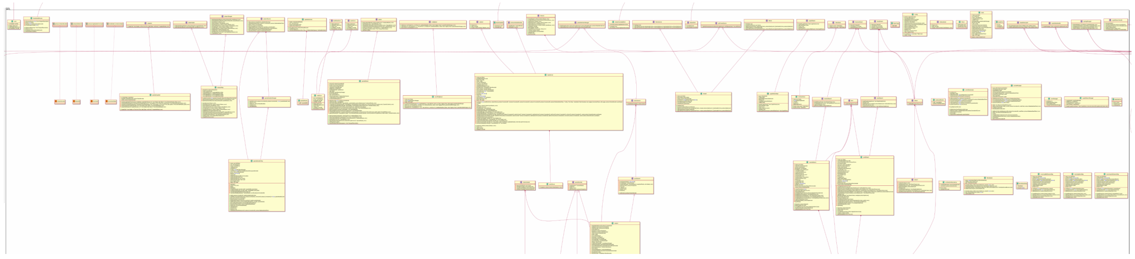


Figure 4.1: Part of the class diagram of the QUIC protocol, made with the plantuml tool [21]

The main reason for creating a new protocol is to make the application as adaptable as possible, so it can easier be used in many situation, as described in section 3.2.
The multicast server is adaptable to output as multicast, but also needs the ability to send the packets to the Rely RLNC encoder, that needs to be running on the same server.

Multicast server

The server imports the protocol takes commandline input to when starting the server it is assigned the network parameters such as multicast address, bind address, network interface and file folder. The server joins the multicast group and start listening on the bind address, by default just the local machine address *127.0.0.1*. The multicast address is needed due to how multicast addresses automatically assigned, but rather picked from which addresses are allowed by the network equipment, in a given address space. For our project the address *224.42.42.1* is used, this is within the allowed multicast address space [4]. The file folder contains the video manifest and segments, it can contain anything, but only the video segments will be transmitted by multicast.

Multicast gateway

The Multicast gateway purpose is to receive the Multicast stream on the managed network, and forward the stream to the end user device. For this to be done in the PoC the gateway listens for HTTP/2 request from end devices. When an incoming request is received, the gateway will first check if the file exists in cache, whether its a request for an index file or a video segment. If the file exists it can serve it right away, if not, it has to request it from the Multicast server, and first after receiving it, the file can be served to the user. This is shown in figure 4.2.

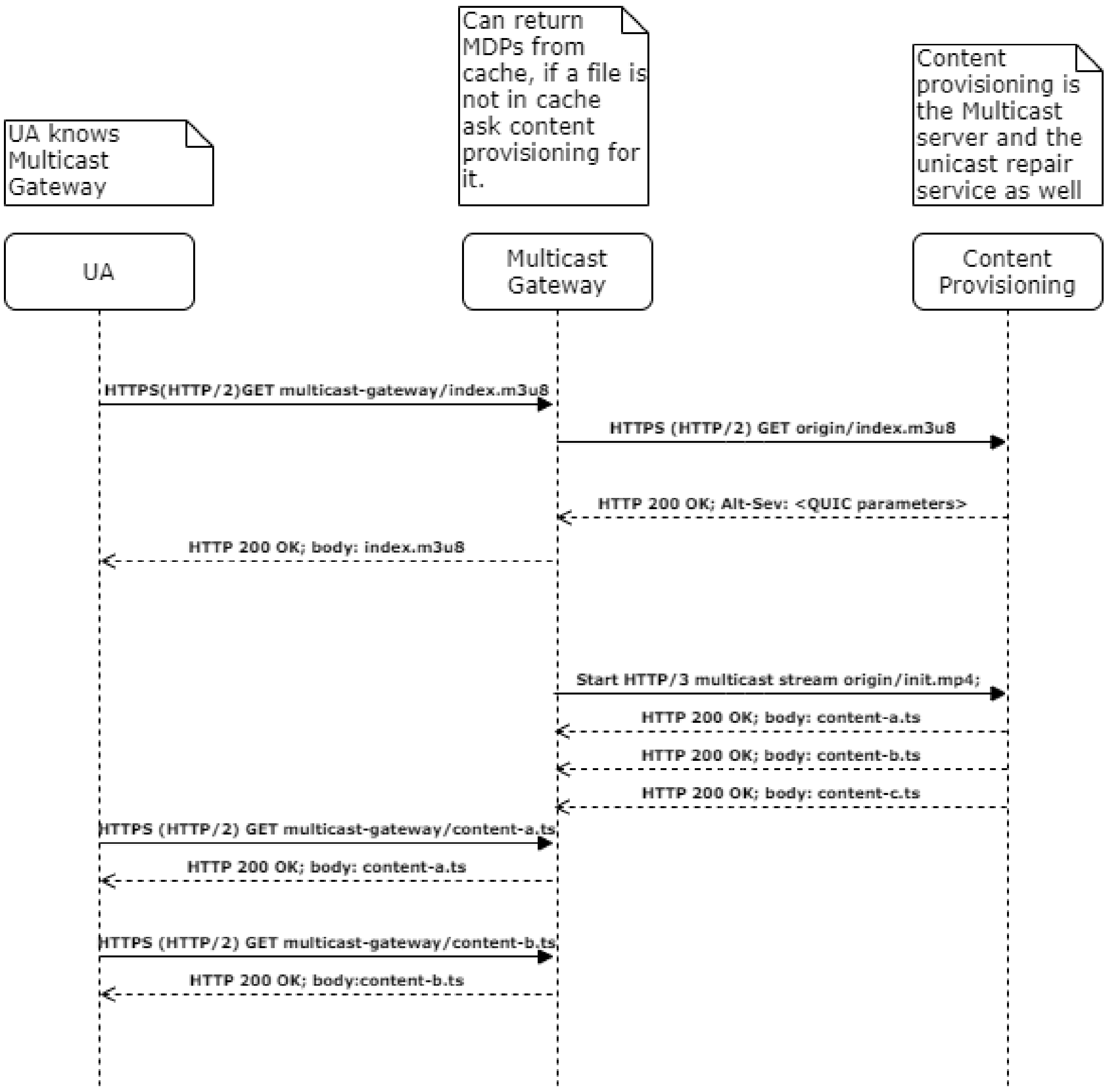

UA knows Multicast Gateway
Can return MDPs from cache, if a file is not in cache ask content provisioning for it.
Content provisioning is the Multicast server and the unicast repair service as well
UA
Multicast Gateway
Content Provisioning
HTTPS(HTTP/2)GET multicast-gateway/index.m3u8
HTTPS (HTTP/2) GET origin/index.m3u8
HTTP 200 OK; Alt-Sev: <QUIC parameters>
HTTP 200 OK; body: index.m3u8
Start HTTP/3 multicast stream origin/init.mp4;
HTTP 200 OK; body: content-a.ts
HTTP 200 OK; body: content-b.ts
HTTP 200 OK; body: content-c.ts
HTTPS (HTTP/2) GET multicast-gateway/content-a.ts
HTTP 200 OK; body: content-a.ts
HTTPS (HTTP/2) GET multicast-gateway/content-b.ts
HTTP 200 OK; body:content-b.ts

Figure 4.2: Flow of first connection PoC, created with diagram.net [9]

In figure 4.2 the first step of requesting the index file, will initiate the gateway to start listening to the Multicast stream, for caching video segments, expecting the end user to start requesting them.

Chapter 5

Evaluation

In this section the testing environment and configuration is introduced.

5.1 Setup

In this section the testing setup, scenarios, configuration and result are explained and presented. The intention of the experiment is to show the difference in using RLNC encoding on both a satellite connection, and a in house network with multicast.

To simulate packet loss, a tool was created to proxy the UDP packets from Rely, then introduce packet loss before being the uplink to the satellite. This is to simulate worse network conditions that are present, due to the satellite conditions being very good during the testing. A UDP proxy tool was programmed to create packetloss after Rely encodes the multicast packets. The testing setup can be seen in figure 5.1.

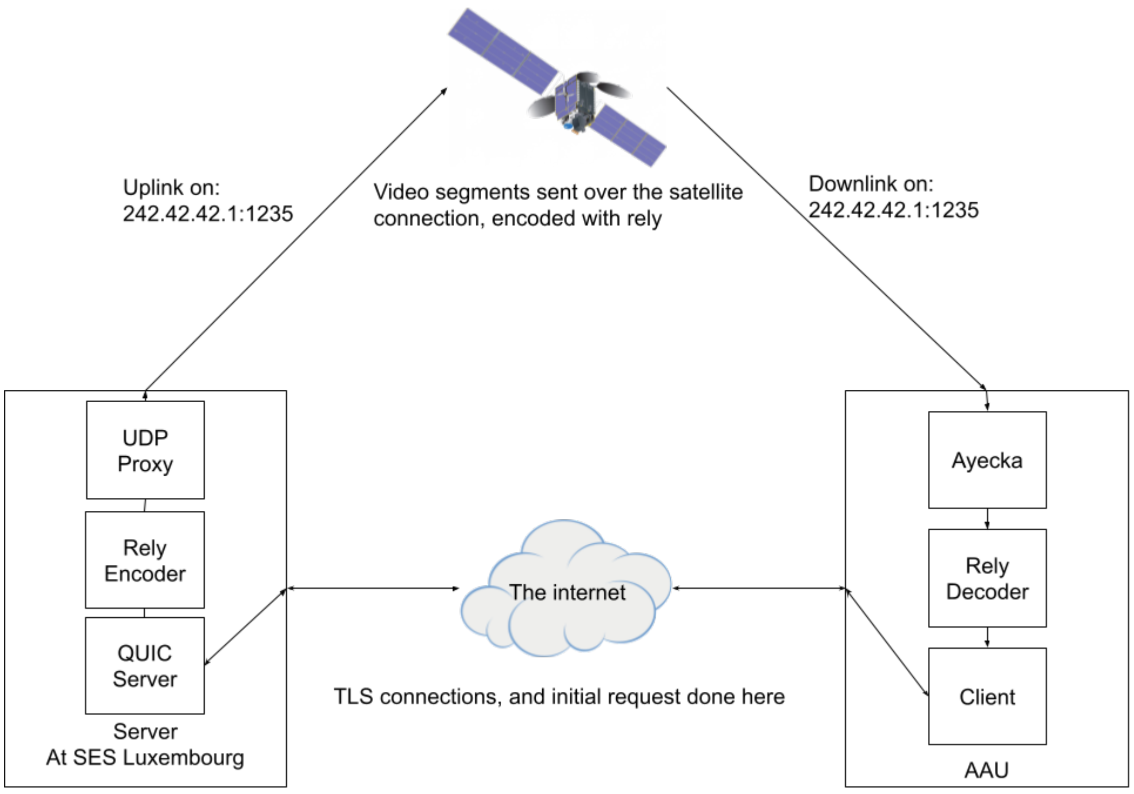


Figure 5.1: Test setup with satellite connection.

Tests on local WiFi network where also done, to get a sense of how effective the RLNC encoding is, in good network conditions, and to test normal WiFi router handeling of multicast. The setup for the local testing can be seen in figure 5.2.

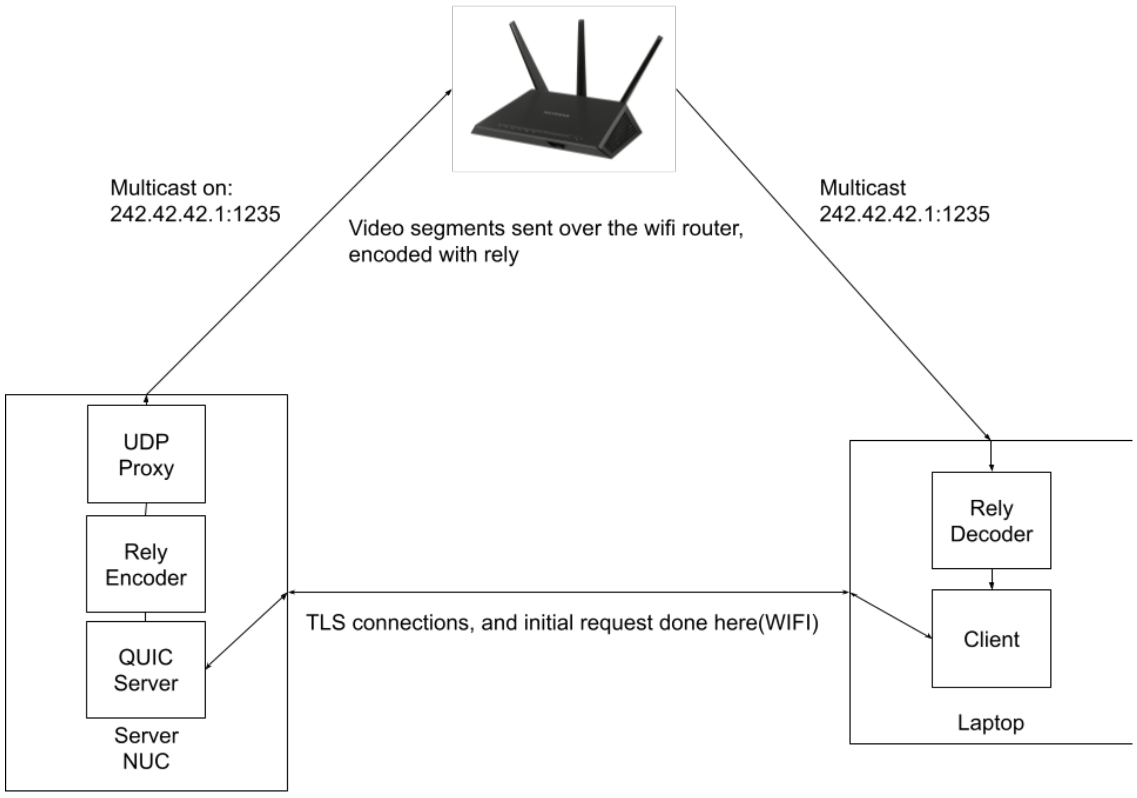


Figure 5.2: Test setup on local WiFi network.

In both test scenarios different configuration are tested, in all the the same video segments and index are used, so the result can be compared evenly later. Both the test on multicast and local network are tested with a delay of 1 second between each segment request.
In each test case multiple Rely configurations are used, according to recommendations from Steinwurf APS, with both 0% and for 10% packet loss. The test done on the local network was a pure QUIC unicast test, QUIC multicast with out Rely and QUIC multicast with Rely repair target to 1 and repair interval set to 5 and 10 respectfully.
The percentage loss from using the UDP packet loss tool, is on top of what the network could potentially lose, this is especially apparent on WiFi network due to multicast support being poor on most standard routers WiFi network [16].

5.2 Result

In this section the result from the testing are presented, and discussed. The intention of the experiment is to show the difference in using RLNC encoding on both a satellite connection, and a in house network with multicast, following the setup in section 5.1. The packet loss tool was essentially created for the satelite connection due to the satellite conditions being so good during the testing, that there was no packet loss. Thus requiring the packet loss tool to simulate worse conditions. The testing setup can be seen in figure 5.1.
The results from the satelite comparing different rely configurations can be seen in figure 5.3.

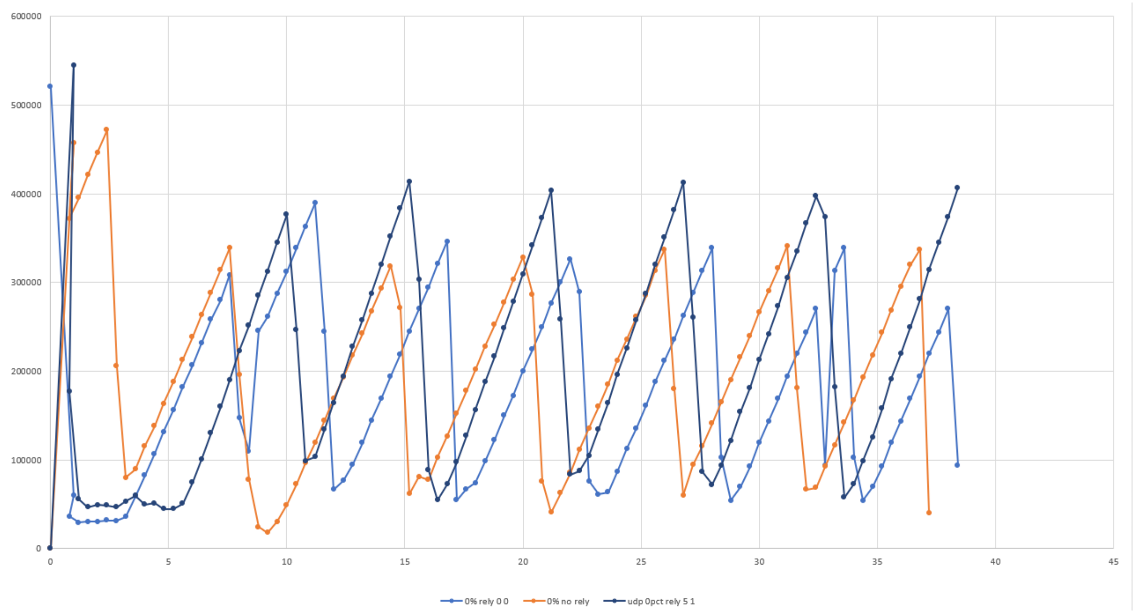


Figure 5.3: Sample of Rely with different configurations on satelite.

The results from figure 5.3, shows the difference in speed and data overhead, each Rely configuration results in with 0% packet loss. The different configurations from right to left are as follows, no rely on, so just the multicast server sending directly to the client via satelite. The next one is with Rely, but without any repair target or interval, this is interesting because it shows the added latency that comes compared to the test with no Rely. The last one is with rely with a repair interval at 5 and repair target at 1, and it shows a clear data overhead, but no latency compared to rely without any configuration.

The test with packet loss are a bit more interesting, first comparing with 10% loss with and without RLNC encoding this can be seen in figure 5.4.

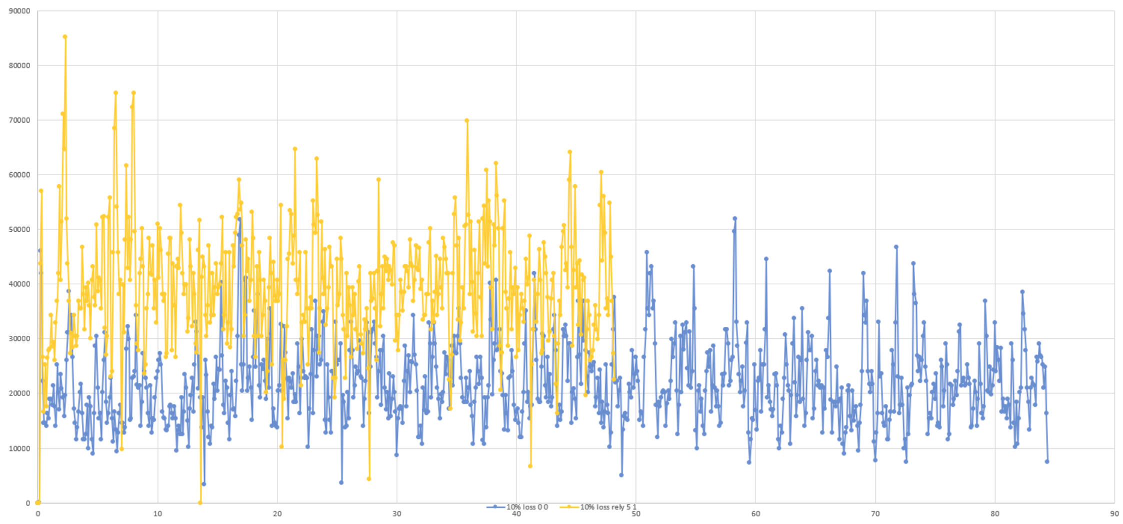


Figure 5.4: Packet loss with and without Rely on satelite.

Figure 5.4 shows how much it helps to have encoding as it is almost twice as fast to transfer the same amount of data. This more interesting finding is that maybe the unicast repair is not optimal when so much packet loss is present. So to understand this better more testing was done on local network shown in figure 5.5, in order to have more controlled environment.

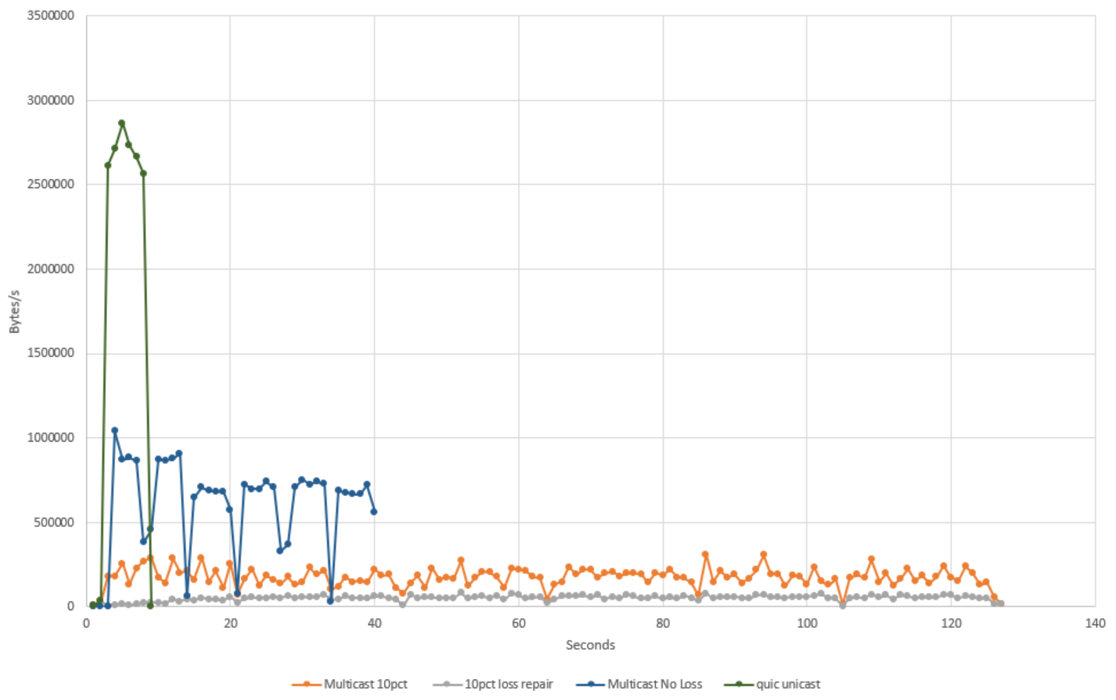


Figure 5.5: Different test runs on local WiFi network.

The test done in the figure 5.5 on the local network was a pure QUIC unicast test and QUIC multicast both without Rely, but one multicast test with 10% packet loss. The most contrasting results from these test where, how fast QUIC in unicast only is compared to everything else. By analysing the data from the multicast test, it is very clear that when packet loss is present, the unicast repair is quickly overwhelmed, the gray graf in the figure, is the unicast repair packets. Resulting in being 3 times as slow compared to multicast without loss, the unicast speed vs the multicast also highlights a flaw that is present in almost all home routers. The poor performance of multicast packets, being limited to a certain amount of packets per second [16]. The natural packet loss on this particular WiFi network with multicast is around 1% which the unicast repair handles quite good, and confirms that with high packet loss, the unicast repair function is lost. Another interesting comparison is how much difference does the rely configuration matter, with the same packet loss, in figure 5.6 the different recommended configuration are shown.

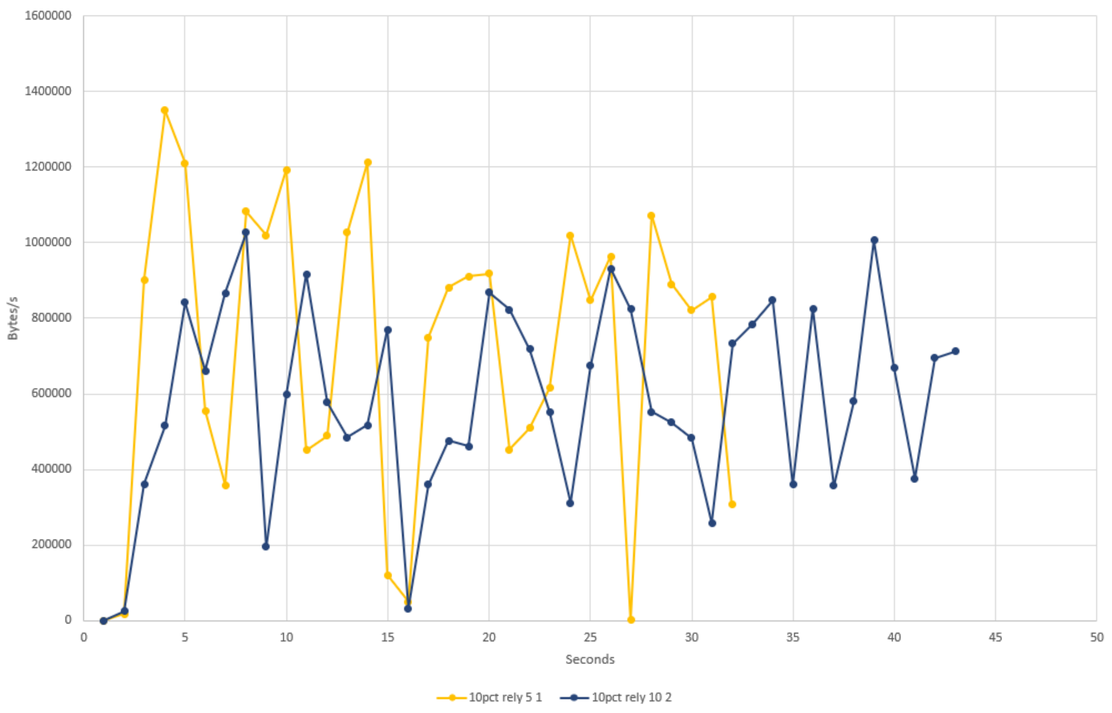


Figure 5.6: Rely at different configurations on local WiFi network.

This shows how the repair interval of 10 and repair target of 2 is slower to transfer when compared to a repair interval of 5 and repair target at 1. On the satelite connection there was no difference between the two, so the local run is probably due to the previously mentioned poor performance of multicast on standard routers [16].

A summary of all the result can be seen in table 5.2, it shows the different configurations, and there most important stats, which is time to first segment, as this is when the user can start the video, as well as the whole transfer time for this test.

heightConfig	Time To First Segment	Total Time In Seconds
0% packet loss, with unicast QUIC local network	0.3s	5.8s
0% packet loss, with multicast QUIC local network	4.5s	36.6s
10% packet loss, with multicast QUIC no RLNC local network	17.7s	124.0s
10% packet loss, with multicast QUIC with RLNC(5,1) local network	3.8s	29.0s
10% packet loss, with multicast QUIC with RLNC(10,2) local network	5.3s	40.0s
0% packet loss, with unicast QUIC SES(internet)	0.7s	11.9s
0% packet loss, with multicast QUIC Satelite	6.6s	38.4s
10% packet loss, with multicast QUIC no RLNC Satelite	14.9s	88.4s
10% packet loss, with multicast QUIC with RLNC(5,1) Satelite	6.7s	48.2s

Table 5.1: Summary of test results.

From the table 5.2, it is very clear that the Rely RLNC encoding is very beneficial in conditions where packet loss might occur, it adds a around 15% overhead in data but cuts the transfer time in half. If the network conditions are better it might be beneficial to use a higher repair interval, to decrease the overhead. Though the test really showed that unicast repair is not very efficient when a high packet loss is present.

Chapter 6

Discussion and Future Work

The project definitely has many areas that need improvement for future implementation regarding stability, features and security. All part of working with a novel and complex technology, that are evolving quickly, so some features that could be interesting to look at in the future are. Being able to fine tune the multicast broadcasting very precise, so if some channel or stream is peaking in viewers, the multicast will prioritize that stream. This in turn would require implementation on both client and server side, to switch quickly, and potentially switch off the active stream for others. These controls could be used for other features such as, stopping certain multicast transmission if there is too much packet lost, this was very apparent that when no RLNC is present the unicast repair is quickly overloaded and slow, as shown in the test results 5.2.

In addition to fine tune the multicast control, the bandwidth control could also be very interesting to investigate, because it could be synchronised with the media encoder. This would enable a new level of adaptive bit rate streaming, as there are more factors than resolution that could improve the quality of experience to the end user. As seen for traditional flow TV where popular channels get more bandwidth [30]. This would enable very high quality video for the most popular live events, if nothing else was required of the multicast network.

It would be very interesting to test the setup in more scenarios, and see if the theorized adaptability keeps true, on all IP supported networks. Also be able to test very high bit-rate streaming on a multicast network able to handle 4k or even 8k video streaming. The scalability of the setup would also be very interesting to test, in cases where many gateways are used, as well as with many end clients. Testing on moving clients, would be interesting to see if there can be handover problems,

and how to solve them, in cases like mobile clients in cars, trains or even planes.
Other applications such as more CDN file distribution could also be an interesting case to investigate further, also from for a more business application point of view. For future security features it would be interesting to build an api gateway to handle the QUIC traffic, and use the authentication tokens to control what stream can be accessed. A feature for the Multicast Gateway is local discovery mechanisms, so devices on the network will recognise, and websites or services supported by it, can start a stream through the gateway. Bringing support to all devices that are able to stream video online, another feature not tested enough are 0-RTT, that allow the Gateway to skip the initial tls handshake. Depending on the network it might be very beneficial for reconnected to an already authorized stream.
The ability to synchronize the video encoding segments with the QUIC packets to the RLNC encoding, would be beneficial for maximizing the reliability of the transfer. So that if packets are lost beyond repair, in for instance a burst of packet loss, only one segment would be affected. Combining this knowledge with the fine control of the multicast stream, a whole video segment could be retransmitted using the multicast stream instead of doing a bunch of unicast repair request.

Chapter 7

Conclusion

The project goal was to implement and showcase a multicast QUIC solution that works reliably on any IP network by utilizing RLNC encoding to ensure reliable transmission on any IP network, utilizing the convenient features of QUIC with built-in security, reliability and the retransmission features. Using the multicast stream for the video transmission on a managed network, which can be any IP supported network, this allows for very efficient content distribution compared to unicast, when many viewers are streaming the same video. The project really showcased the strong suits of using RLNC in section 5.2, but also exposed the problems for unicast repair, when packet loss in apparent. This is then an interesting showcase of how much the RLNC encoding really improves the reliability, especially in networks where retransmission is expensive or slow. The sections 3 and 4, outlines the requirements for creating a system that can reliably deliver content using multicast QUIC, and the test results in section 5.2, reveal the abilities, and pitfalls of the solution. Which goes to show that there is room for improvement, but still being able to handle 10% packet loss without any major issues. Therefore the research question "*How to implement a reliable multicast solution for content delivery using QUIC on any IP supported network?*", is considered answered which concludes this project.

List of Figures

List of Tables

Bibliography

[1] "2020 super bowl streaming audience up 23% to 7.4 million", [Online]. Available: https://nscreenmedia.com/2020-super-bowl-streaming-audience/.

[2] J. H. Aleksander Nowak, "Evolution of media distribution with multicast quic", English, 2020. [Online]. Available: http://jonessoftwaresolutions.com/P8.pdf.

[3] Amakai and M. Bishop, "Hypertext transfer protocol version 3 (http/3)", Feb. 2021.

[4] I. A. N. Authority, "Ipv4 multicast address space registry", [Online]. Available: https://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml.

[5] R. Bradbury and L. Pardue, "Linx100: Scalable internet broadcasting using multicast quic", [Online]. Available: https://youtu.be/7l6mXM5CrmQ.

[6] Cisco, "Cisco annual internet report (2018–2023) white paper", 2021. [Online]. Available: 10.04.2021.

[7] L. Clemente, "A quic implementation in pure go", [Online]. Available: https://github.com/lucas-clemente/quic-go.

[8] P. Cluff, "The low latency live streaming landscape in 2019", 2019. [Online]. Available: https://mux.com/blog/the-low-latency-live-streaming-landscape-in-2019/.

[9] Diagram.net, "Diagram with anyone, anywhere." [Online]. Available: https://www.diagrams.net/.

[10] DVB, "Adaptive media streaming over ip multicast dvb a176", 2021.

[11] "Google drawing", [Online]. Available: 24.05.2021.

[12] J. Guan, X. Liu, S. Yao, and Z. Jiang, “Design and implementation of a central-controllable and secure multicast system based on universal identifier network”, English, *Sensors*, vol. 18, no. 7, p. 2135, 2018. [Online]. Available: `https://search-proquest-com.zorac.aub.aau.dk/docview/2108750077?accountid=814`.

[13] D. Hardt, “The oauth 2.0 authorization framework”, Sep. 2012. [Online]. Available: `https://tools.ietf.org/html/rfc6749`.

[14] L.Pardue, R.Bradbury, and S.Hurst, “Hypertext transfer protocol (http) over multicast quic”, Feb. 2019.

[15] W. Law, “Three roads to jerusalem”, 2019. [Online]. Available: `https://www.youtube.com/watch?v=Col12gjnNlI`.

[16] C. P. M. McBride, “Multicast wifi problem statement”,, 2017. [Online]. Available: `https://tools.ietf.org/id/draft-mcbride-mboned-wifi-mcast-problem-statement-01.html`.

[17] B. Moore, “The best sports streaming services for 2021”, 2021. [Online]. Available: `https://www.pcmag.com/picks/the-best-sports-streaming-services`.

[18] E. N.Sakimura, J. Bradley, and N. Agarwal, “Proof key for code exchange by oauth public clients”, Oct. 2015. [Online]. Available: `https://tools.ietf.org/html/rfc7636`.

[19] L. Pardue, “There and back again - reinventing udp streaming with quic”, [Online]. Available: `https://youtu.be/Zdkjd7-EWmQ`.

[20] C. Perkins and J. Ott, “Real-time audio-visual media transport over quic”,, 2018.

[21] PlantUML, “Plantuml in a nutshell”, 2021. [Online]. Available: `https://plantuml.com/`.

[22] J. Porter, “Youtube joins netflix in reducing video quality in europe”, [Online]. Available: `https://www.theverge.com/2021/3/20/21187930/youtube-reduces-streaming-quality-european-union-coronavirus-bandwidth-internet-traffic`.

[23] “Random linear network coding (rlnc)-based symbol representation”, [Online]. Available: `26.05.2021`.

[24] Steinwurf, “Low latency and reliability for transport”, [Online]. Available: `https://rely.steinwurf.com`.

[25] Steinwurf, “Steinwurf technical documentation”, [Online]. Available: `http://docs.steinwurf.com/`.

[26] D. Stenberg, “Http/3 - http over quic is the next generation”, [Online]. Available: `https://youtu.be/idViw4anA6E`.

[27] D. Stenberg, “Http/3 explained”, [Online]. Available: `https://http3-explained.haxx.se/en`.

[28] W. T. W. T. Surveys, “Usage statistics of http/2 for websites”, [Online]. Available: `https://w3techs.com/technologies/details/ce-http2`.

[29] Techsawa, “Understanding satellite tv encryption”, 2019. [Online]. Available: `https://www.techsawa.com/free-to-air/understanding-satellite-tv-encryption/`.

[30] Tektronix, “A guide to ensuring digital video service quality”, English, 2014. [Online]. Available: `https://www.telestream.net/pdfs/general/Guide-to-Ensuring-Digital-Video-Service-Quality-25W309040.pdf`.