A study on utilizing the WebRTC P2P protocol to reduce infrastructure costs of media streaming

B2268248

Jan Ivkovic

A thesis submitted
in partial fulfillment of the requirements for
the degree of Master of Arts
in the Sophia University
Graduate Program in Global Studies

MENTOR: **Prof. Yacob Khojasteh**

READER: **Prof. Peter De Maeyer**

READER: **Prof. Seyed Mohammad Mousavi Jahan Abadi**

January, 2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

LIST OF ABBREVIATIONS

ISP – Internet Service Provider

WebRTC – Web Real-Time Communication

HTTP – Hypertext Transfer Protocol

HLS – HTTP Live Streaming

RTMP – Real Time Messaging Protocol

RTSP – Real Time Streaming Protocol

NAT – Network Address Translators

STUN – Session Traversal Utilities for NAT

TURN – Traversal Using Relays around NAT

LL-HLS – low latency-HLS

OBS – Open Broadcast Software

ABSTRACT

In the fast changing and rapidly developing digital environment, transmitting data over the internet has become one of the key pillars in building online businesses and platforms that provide users with unique experiences and solutions that improve and simplify the everyday lives of internet users. As such it will come at no surprise that the number of users, as well as the sheer amount of data transmitted every year through the internet has been increasing for years, since before the dot-com bubble (Odlyzko, 2003) and into the present with the recent spike due to the Covid-19 pandemic and work from home, which required workers to use conferencing software to work together on projects remotely (Feldmann and others, 2021). This rapid increase in the volume of data being transmitted of course means that ISPs have to upgrade their infrastructure to be able to cope with the ever increasing demand from clients, but the same is also true for many businesses and services, which – in order to stay competitive and provide modern services – must stream vast amounts of data through servers and other infrastructure, which they either have to built themselves or rent from other vendors, that can considerably increase the costs associated with providing a service. This is especially true for businesses and platforms that provide streaming as a main part of their service, such as video sharing and live streaming platforms, file sharing platforms and the like.

However, with the rise of modern internet protocols and standards, businesses have more choices in how to manage and distribute data across networks, allowing them to opt to utilize modern concepts and paradigms, which were not utilized or available prior

and thus manage and lower their expenses better. An example of this is Peer to Peer (hereafter referred to as P2P) networking, which allows businesses to bypass using the traditional peer to server to peer model, which requires servers to manage the streaming of data and allows the users to stream information amongst themselves almost directly depending on the network configurations and other conditions. This modern approach could be used as a new disruptive alternative to the tried and tested method, allowing companies to cut costs or provide new types of services.

The main objective of this thesis is to assess the use of Web Real-Time Communication (hereafter referred to as WebRTC) – a fairly standardized open source P2P framework, supported by most browsers and programming languages/frameworks – when it comes to the live streaming of data as compared to the traditional, tried and tested methods – such as HTTP Live Streaming (hereafter referred to as HLS). Specifically the main focus is on video streaming across the P2P network, as video streaming is one of the most resource demanding types of data streaming on the internet. This thesis evaluates the use of this technology in terms of viability, quality, as well as complexity and the challenges faced by implementing this new model.

1 INTRODUCTION

In order to make the analysis and draw conclusions from the comparisons of the traditional and aforementioned proposed models, we must first overview and analyze the technologies involved, in order to understand their fundamental characteristics, limitations, challenges and way of implementation into real world applications. It is also important to define the specific characteristics we are looking for, which can better

narrow down the choice of technology used and further explain the methodology in the next chapter.

As the methodology of researching requires the use of a benchmark, this chapter first briefly presents the “traditional” ways of streaming videos online, which gives a better understanding of the nuances and differences between other internet protocols/frameworks and WebRTC. It also presents the WebRTC technology and architecture that encompasses it and all the various applications used in the test to ensure the connection is established and maintained. Some of the systems used in the testing are open source projects used for the simplicity of integration, while others are custom made for the purposes of the research. This chapter only covers the open source projects in depth, while the custom made applications and systems are further explained in the methodology.

2 LITERATURE REVIEW

As analyzed in Internet traffic growth: sources and implications (Odlyzko, 2003) and A Year in Lockdown: How the Waves of COVID-19 Impact Internet Traffic (Feldmann and others, 2021) internet usage has been growing since its creation with no signs of slowdown especially as it became even more prevalent during the Covid-19 pandemic which saw an increase of usage by companies and educational institutions. This fact is the basis of this research which acknowledges that the increase will continue and thus the need for infrastructure and better techniques for managing communication will also continue.

In A scalable load generation framework for evaluation of video streaming workflows in the cloud (2020) Ramos-Chavez and others present a study on how to measure loads on the traditional streaming infrastructure that utilizes DASH or HLS. Their study focuses on creating a testing suite that can generate clients that simulate load on an infrastructure used for testing infrastructure for deployment. This research builds upon this concept of testing by creating a testing suite for the WebRTC protocol and uses the research as a basis point and comparison between the two methods of streaming.

The impact of bandwidth limitations and video resolution size on QoE for WebRTC-based mobile multi-party video conferencing (2016) by Vučić and others presents the limitations of WebRTC in terms of streaming between peers in the standard conference call type of application. The study compares streaming at resolutions which are smaller than what would be considered optimal for streaming purposes – the highest resolution is 960 by 640 pixels. The study was done under controlled network conditions which might not accurately represent the true limitations in a real world scenario, however, it does show that the protocol does have limitations on the client side which could be further explored.

3 PROBLEM DESCRIPTION

The main goal of this research paper is presenting and evaluation a new, alternative streaming model to the “traditional” well established one, which is currently in use by most companies and major online platforms. As such the research presents a test architecture, which utilizes the WebRTC protocol, along with custom built applications

and systems deigned for testing the model. The main points of focus of this research are three main questions:

1. Feasibility – whether the proposed model can be set up with the existing P2P technology,

2. Performance and efficiency – whether the proposed model is actually a better alternative to the existing one as it would otherwise make no sense to pursue utilizing it,

3. Scalability and limitations – whether the limitations in scalability or elsewhere would make it impossible or difficult to deploy a large scale application or platform, utilizing the WebRTC P2P system, in production.

3.1 The "traditional" models

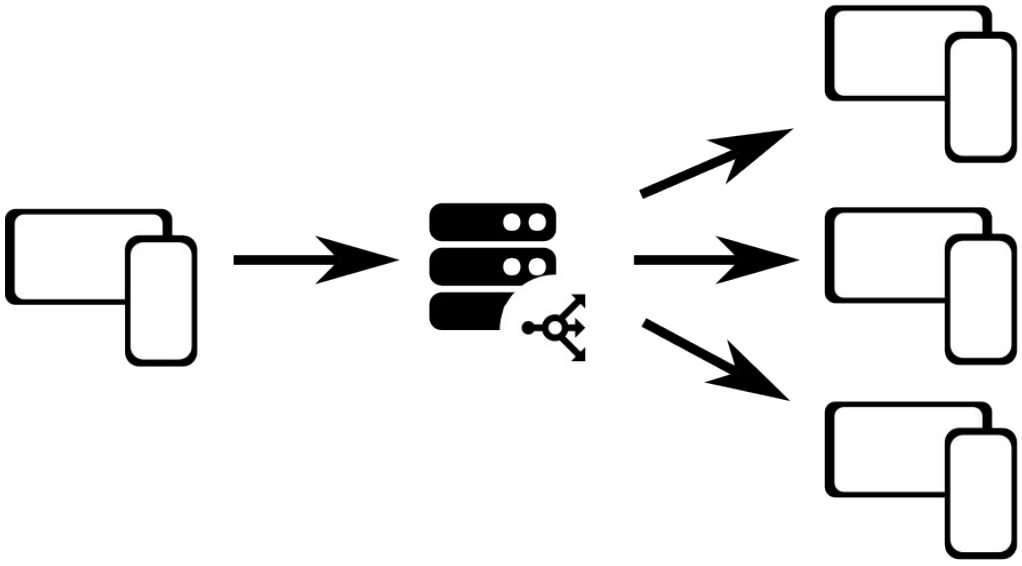
Streaming has a long history spanning decades from the 1990s, when the first internet streaming services and protocols were developed, allowing for the creation of platforms such as YouTube and Netflix. Over the years there have been many different protocols used for different purposes and with different goals and thus characteristics. Many of the standards were never widely used or adopted, however, among the ones which were, the most notable are Real Time Messaging Protocol (hereafter referred to as RTMP), Real Time Streaming Protocol (hereafter referred to as RTSP), and HLS.

When choosing the appropriate protocol for testing, it is worth keeping in mind the main goal of this research, which is to present an alternative P2P solution utilizing WebRTC to using the most used protocols used for streaming today. With this in mind,

a very important characteristic of the chosen protocol must be that it is an industry standard, chosen for its performance, stability, availability, and reliability.

Irrespective of the choice between the three presented options, however, they all utilize a similar kind of network topology as they all would have to utilize some sort of server, which would act as a sort of relay between the streamer client – the client creating and streaming the content or video –, and the subscriber clients – clients which would "subscribe" to a stream and continuously download, process and display it to the user. Therefore, this model requires a use of a server or multiple servers, which maintain the connection and relay information between users, which can be quite costly for companies operating such platforms, which is what would ideally be replaced by the alternative. The Figure 1 below visualizes the traditional model.

Figure 1: The "Traditional" model of streaming using a server



3.1.1 RTMP

RTMP is a protocol, which was developed and primarily used for high performance transmission of audio, video, and other data between Adobe Flash platforms, such as the Adobe Flash Player and Adobe AIR (Nunez and Toasa, 2020). At a certain point in time it was considered as being the industry standard in many use cases; however, with the Flash Player no longer being supported by Adobe after reaching the End-Of-Life on December 31, 2020, the use of this protocol has sharply declined, making it a viable, but less likely choice for big platforms and services (Adobe, 2021).

3.1.2 RTSP

RTSP, unlike RTMP and HLS, typically operates utilizing the UDP protocol, making it more comparable to WebRTC, which does the same (Nunez and Toasa, 2020). However, RTSP is not as commonly used as it has a worse performance and quality than HLS, because it does not support adaptive streaming and is prone to packet losses, making it prone to connection errors, sudden disconnection issues and jittering. Another issue it has is the lack of support by most browsers and on iOS devices, making it more difficult to set up and use for most platforms without workarounds.

3.1.3 HLS

As opposed to the previous two solutions, HLS is widely supported by most browsers today and has good support on all devices. It was developed by Apple and has become widely adopted by many corporations and platforms, which opt for this solution as it is

a lot more reliable and generally produces a lot better results in terms of performance and user experience (De Cicco and Mascolo, 2014).

It utilizes the HTTP protocol for transportation of streams that are broken down into segments and then put together into a buffer on the client device. This makes the stream more reliable, but the need for buffering means that transporting a “live” video and audio feed requires extra time as enough frames have to be gathered for the following segment and then have to be joined at the client device, resulting in a delayed stream that can be as high as 4 to 14 or even more seconds (Durak and others, 2020).

Due to the wide adoption and a good user experience, the HLS protocol seems to be the most suitable as a benchmark as it represents a real world, in production protocol, which is what users would likely compare any alternative to in terms of performance and experience.

3.2 WebRTC

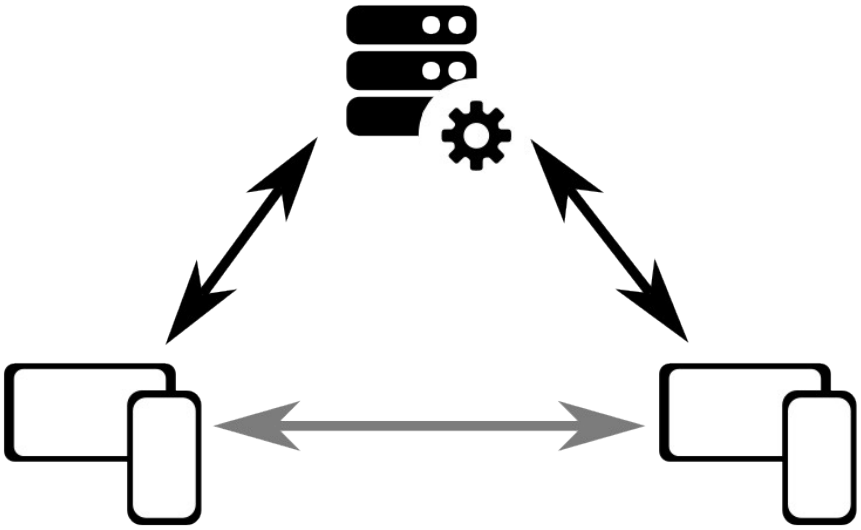
The alternative to using a traditional streaming network topology is the utilization of P2P technology, which in theory means connecting the various end users with one another without the heavy reliability on server resources, though in some cases and exceptions this cannot be perfectly achieved, and thus a workaround or alternative is required in order to insure performance and quality of transmission. This solution represents a sort of mesh network topology, which means that the peers are connected with one another forming a mesh. In certain applications and use cases – such as online conference and video calls – it requires that all the peers have a connection established

with all the other ones, leading to a requirement of maintaining $\frac{n(n-1)}{2}$, where n is the number of peers; however, in case of streaming that number is only equal to the number of subscribers to a stream (Sanni and others, 2013).

WebRTC is the most popular open source framework and P2P solution for real time communication, as it has wide support on most internet browsers, as well as most modern devices and programming languages, making it ideal for most P2P applications. It was developed by Google and has been used by many companies to create modern P2P applications and platforms such as Zoom (Google for Developers).

In essence WebRTC is a framework responsible for establishing and maintaining a connection between the peers which are connected with one another. The challenge of connecting peers on the internet is that each device has a certain IP address which can change over time and with the change in network configuration – such as for example going from a WiFi connection to using 4G or change in the network tower in use. As such the peers need to "find" one another using some sort of third actor, which usually is a sort of signaling server which maintains a log of all available peers and can relay to them the information about other peers and vice versa, which allows two peers to share their network settings and accessibility with one another, establishing the P2P connection. The Figure 2 below shows the schema of two peers having access to a signaling server, which is used so they may share their locations and information, leading to a P2P connection, after which the connection with the server may be closed depending on the use case.

Figure 2: WebRTC peers connecting to a signaling server and establishing a P2P connection with one another



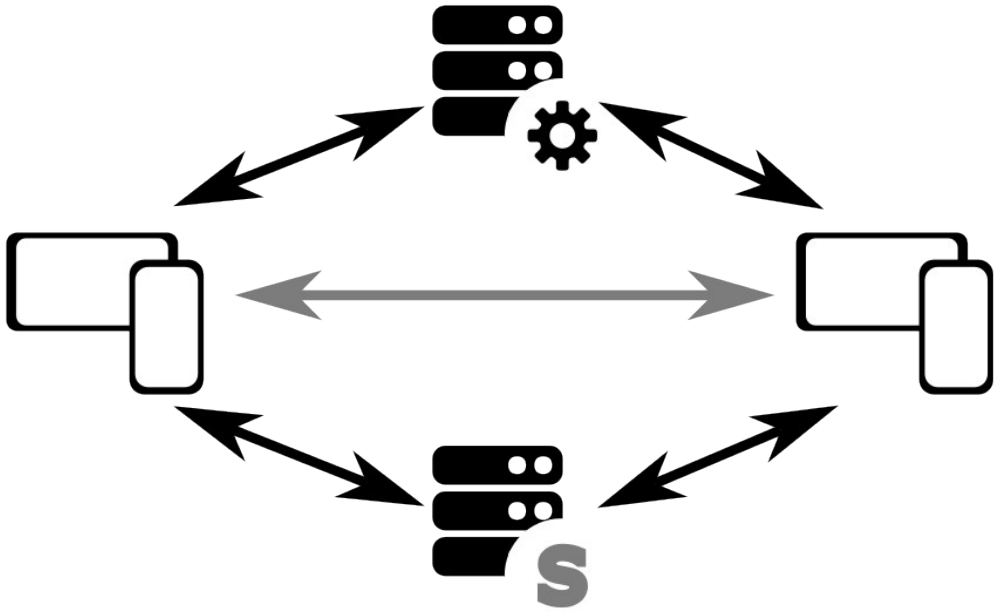
3.2.1 STUN server

One of the core concepts and parts of WebRTC is Network Address Translation (hereafter referred to as NAT), which is the solution to the aforementioned problem of connecting two peers online, as they both have to find out their own network address and share it with one another. Although the signaling server is responsible with organization of peer connections – sometimes referred to as lobbies – and with sharing the initial data between two peers until a direct P2P connection is established, this server is typically not responsible for gathering network information about the peers, which they need to share in order to establish a P2P channel. The server responsible with NET traversal is called a Session Traversal Utilities for NET server (hereafter

referred to as a STUN server). It is a core part of the architecture, as it has to be accessed by each client so that they may receive their own network address, which is later shared with each peer they intend to establish a direct connection with (Vegibit).

The Figure 3 below is a representation of the connections between two peers which are trying to establish a connection, as well as a signaling server and a STUN server which are required for establishing the connection. This figure is all of setup required for WebRTC to work in ideal conditions with client applications running on the two devices.

Figure 3: WebRTC architecture with a signaling server and a STUN server



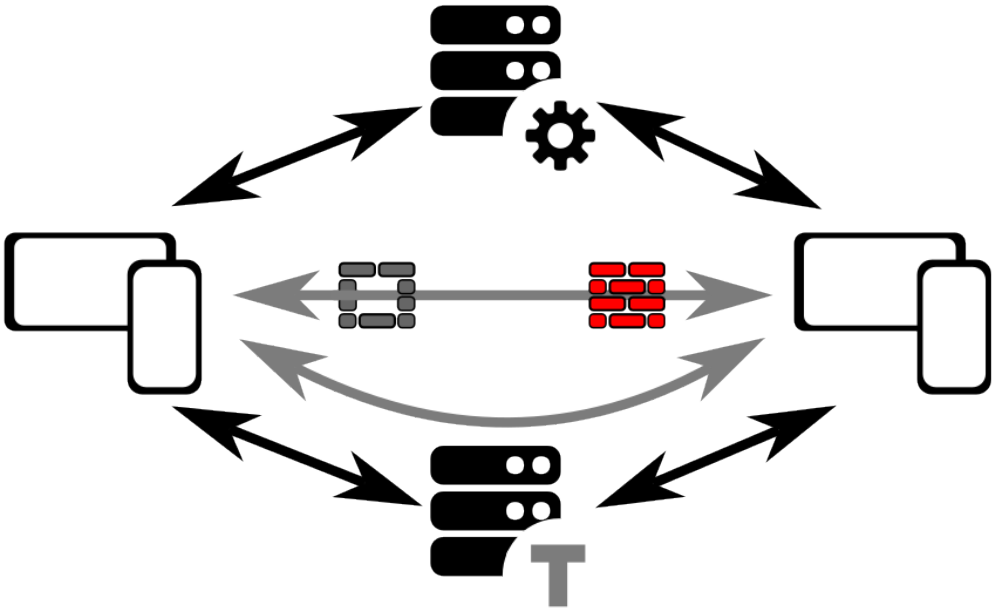
3.2.2 TURN server

Though WebRTC can function just by utilizing a signaling server, STUN server and client side applications, it does not function in all circumstances. This is due to the fact that there are many different types of network configurations, each of which has very specific characteristics, and many of them do not allow for a direct connection between the peers to be established for security or other reasons, such as firewall or router configuration. Due to this, sometimes there is no alternative other than having a relaying server in between the peers that cannot be connected directly, which simply works by sending information sent from one peer to the other and vice versa. This type of server is called a Traversal Using Relays around NAT server (hereafter referred to as a TURN server). This server has the ability to not only connect two peers directly like a STUN server – by sending a NAT traversal –, but may also be used as a relay server in cases where establishing such a connection is not possible, thus resembling more a server used in "traditional" streaming of data explained above, however, it must be noted that this server typically creates a connection for each two peers that connect indirectly, meaning that it cannot simply relay information from one client to many, but must receive individually a packet from a client for each corresponding client, which causes a lot of redundant information being sent. One of the main reasons behind this is the way streams are encrypted during data transfer between peers (Stream).

The Figure 4 below shows a WebRTC connection where the right peer is behind a firewall which makes a direct P2P connection from the left peer to it impossible, which results in a creation of an alternative route for the stream, which passes through the

TURN server. This is a fairly common scenario in real world platform deployments, making it a common requirement to incorporate a TURN server into a deployment.

Figure 4: WebRTC architecture with a TURN server where the right peer is not accessible due to a firewall and the data is relayed via the TURN server



4 METHODOLOGY

This chapter goes over the methodology and the setup for testing the technology based on the three main questions, explaining in detail the applications used, as well as the ones developed specifically for the purpose of this research and testing. It also touches on the test system design choices which were made while designing the development and test architecture and the reasons for the choices.

The first two subchapters go over the basic setup of the benchmark and the proposed model by explaining the chosen software used and its purposes. This part mainly addresses the feasibility point of the research as the software proposed must be feasible to build and run in the intended form. The third subchapter goes over the software and systems used for doing the network analysis on the servers to check and evaluate the performance of the model and the benchmark and to draw comparisons to the two and evaluate the benefits or drawbacks of the proposed model. And the final subchapter focuses on client side tests which are performed in order to evaluate the user experience, as well as limitations and scalability of the solution.

4.1 Benchmark model setup

As explained in chapter 1.1 The “traditional” models, the most widely used protocol for live streaming currently in use and thus the most direct competitor to P2P is HLS, which is based on HTTP and in most cases divides the video feed into 1-10 second long segments (Durak and others, 2020).

4.1.1 Server

The chosen server for benchmark testing was Nginx, as it is an open source project available on GitHub, making it very accessible for testing and required modification as necessary. The server was set up with a module called Nginx-RTMP, which allows for the transmission of RTMP to the server and later transforms it and hosts it as an HLS stream. The reason for using RTMP for traffic from the streamer to the server is that

although HLS is great for consuming streams, it is not intended to be used for clients to transmit HLS streams as a server style configuration is required with static IP addresses.

Nginx is a web server, that can be utilized as a proxy, load balancer or mail server. It can be used as an entry point into a network or as a standalone server, which is very flexible in options and configuration, making it well suited for this project (Nginx, 2023).

4.1.2 Client side apps

When it comes to client applications for the benchmark setup, the benefit of using the HLS protocol is that the only thing required for the access and downloading of the stream is any modern web browser, making it very easy to test and quickly visualize the results of a stream. However, other applications like VLC media players are also able to access the stream without the requirement for a browser (Wilbert, 2023).

For uploading the streams, however, we need to use an application that can transmit RTMP such as Open Broadcast Software Studio (hereafter referred to as a OBS Studio). This software is capable of streaming both video from a camera, as well as screen capture, making it as versatile and comparable to the WebRTC option (OBS Project, 2023).

4.2 WebRTC model setup

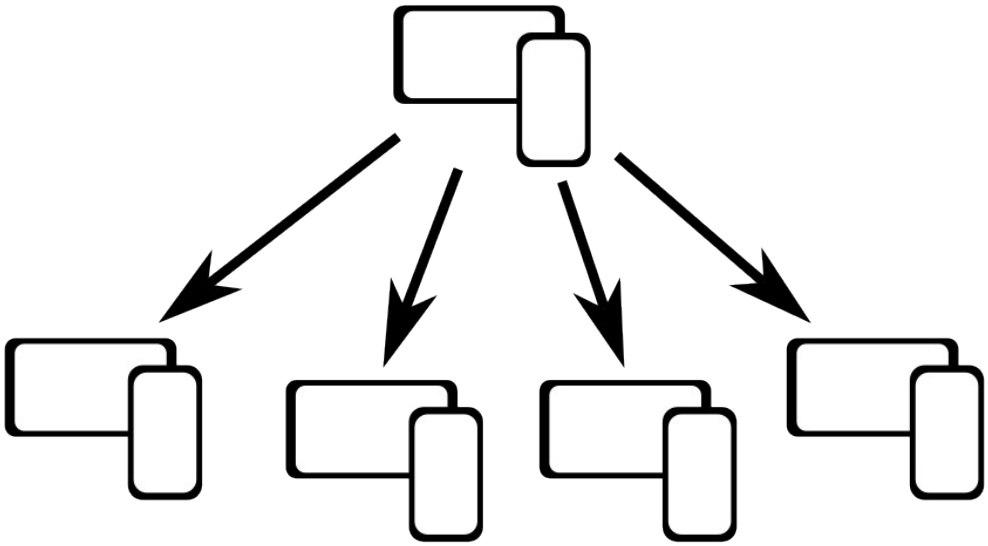
The WebRTC protocol is quite complex with many variables and exceptions as explained in section 1.2 WebRTC, making it sensible to implement an architecture that is fairly complex and that covers many edge cases, which would likely most accurately

represent the in production environment of such a complex system. This means that one should opt in to using a full TURN server setup, as opposed to only a STUS server to insure stability and quality of media being transferred. At the same time it is reasonable to assume that a TURN server would likely be the most highly taxed part of such infrastructure, as it is the one responsible to essentially proxy or relay information between peers and has to run at all times, as opposed to a signaling server for example – that only runs during the initial P2P connection establishment phase of the WebRTC protocol –, which would likely give a more real world and accurate measurement of server performance.

4.2.1 Signaling server

The architecture of the proposed system requires a specific use case with specific characteristics in the way peers are connected with one another. The most common topology for models utilizing WebRTC is a sort of mesh topology, where all pears are connected to all the other peers, as that is the desired configuration for applications like conference call platforms; however, as we are testing streaming where the peer connection topology is one to many, this mesh structure does not make as much sense, as it would only be an additional processing overhead. A more desired topology would be a sort of tree topology where multiple clients would subscribe to a stream of one streaming client as visualized on Figure 5 below.

Figure 5: WebRTC peer tree topology



The signaling server would therefore be required to first establish a connection with a client application and then either allow the client to be upgraded to a streamer, preventing the connection to the server from timing out and being registered as a streamer, or be provided with available streamer clients to which the client would be able to connect as a subscriber to a client stream, which upon completion of the P2P exchange would result in the subscriber client disconnecting from the signaling server. This setup is a bare minimum, simple solution to handling client connections in a way that only connects clients that need to be connected and disconnects clients from the server that no longer need that connection after setup is finished to save resources.

As this kind of setup is very unique and to the best of my knowledge no such setup exists, at least in a ready to use form, it was most sensible to develop a custom made server in Go, as the language is very suitable for such applications.

4.2.2 Coturn

As the WebRTC model also requires a TURN server, Coturn was chosen as the server providing this functionality. The reason for this choice is that Coturn is a free and open source implementation of a STUN and TURN server, that is one of the most popular amongst available options and provides many features and flexible configuration. It also offers support for transport layer security (hereafter referred to as TLS) which is required in order for some platforms and browsers to be able to establish a connection (Coturn, 2023).

4.2.3 Client app

There are many options for client applications, as the WebRTC protocol is wisely used and supported by many platforms and browsers. It would be possible to create a native application for each platform – Android, iOS, PC –, but it is a lot simpler to simply create a web application, which basically is just a website containing JavaScript that supports Web-sockets and WebRTC.

For the sake of simplicity the chosen approach was crating two websites, where the main system would be Javascript files handling the connections to the signaling server, TURN server and the P2P connection when established. The reason for using two websites is so that one would act as a streamer, requesting to be upgraded by the

signaling server and sending streams to the other type – subscribers – displaying the stream visually. It would of course be possible to only have one web application supporting both functionalities with UI options, but to keep things simpler the divided approach was chosen.

4.3 Network traffic testing

In order to be able to assess the performance and efficiency part of the research, network has to be analyzed for comparison. This has to be done on the servers running the various server applications required for both the "traditional" method and WebRTC method, as this is where cost management measures could be applied by platforms and companies. It is also worth mentioning that though measuring the load on processors and memory could also be performed, it would likely not be as directly correlated to the cost of network infrastructure as most companies deploy specialized hardware at scale, which in case of this sort of networking would specialize on the network speeds rather than processing, because the main cost of that hardware is in the networking components.

An important consideration when doing the testing of network loads is considering that although the "traditional" model in our example is only a single server, which means we only measure network load on that server, the architecture is more complicated in the proposed alternative model. In the case of testing network load when using WebRTC, one solution would be to look at network load on the signaling server and on the TURN server and sum it up for a particular period of time, however, that would be difficult and prone to errors. However, it is possible to overcome this issue by running both the

signaling server, as well as the TURN server on the same physical device and just getting the sum of the network load of the processes.

In terms of software used for monitoring network traffic, as the server used for the testing will be a Linux distribution, Nethogs was chosen as a monitoring tool for its ease of use, simplicity and open source nature (Raboof, 2023). To collect the data we use sed, which is a text application for Linux that allows for taking snapshots of the terminal at various intervals, for example every second, which is what we do in this research using the following command:

sudo nethogs -t -d 1 | sed 's/[^[:print:][:cntrl:]]//g' > in.txt

This creates a file called in.txt, which is a simple text file with snapshots of network usage taken every second. The data collected using this software is then later processed using a custom made Go script that eliminates irrelevant data and converts the text file into a csv file which can be further used in spreadsheet and visualization software.

In order to run the test in a controlled and verifiable method, both the HLS benchmark and WebRTC tests must run with dummy clients that connect in the similar fashion as explained in the subchapters.

4.3.1 HLS dummy clients

Though receiving a stream is fairly straightforward when it comes to HLS, for larger scale testing it would be highly impractical to need to open a new browser tab to add a client and it would also produce inconsistent results. Thus for testing purposes, the load generation tool used in the paper “A scalable load generation framework for evaluation

of video streaming workflows in the cloud" (Ramos-Chavez and others, 2020) was used to simulate network load on the server. The software was configured in a way that it added a new user every second until it reached 100 users. This way it would be easier to compare the load on the server at each step of the increase and then compare it to the proposed WebRTC model.

4.3.2 WebRTC dummy clients

For WebRTC on the other hand, no testing suite or application exists to the best of my knowledge. As such it was decided that the best approach would be to build a custom made testing suite application in Go using the Pion library for WebRTC. This approach ensures that the application and in extension the tests can be as comparable to the aforementioned benchmark used in the other research. In essence the dummy testing suite would at specified time intervals initiate a Web-socket connection to the signaling server responsible for creating new P2P connections and would make a request and connect a new client to the streamer.

4.3.3 Experiment specifications

To further clarify the experimentation environment and characteristics, the experiments were conducted for the video resolutions of 640 by 480 pixels, 1280 by 720 pixels, and 1920 by 1080 pixels, for both HLS and WebRTC, where:

- Each second a new dummy client was added using the appropriate test application,
- The dummy clients increased to 100 unless the connection was cut,

- Each second a new snapshot of Nethogs was taken.

4.4 Model limitations testing

In order to both test the feasibility and practicality of the proposed P2P model, it has to be rigorously tested and assessed from a user experience perspective. Simply the feasibility of a P2P model being possible does not make it a viable and good alternative, as it might have performance issues on the user side or have significant flaws or limitations hindering its adoption into full scale in production system and environments. In fact this is the main issue eluded to in tech communities as to why this technology is not being used as an alternative to the traditional model (Vicic and others, 2016).

For the purposes of testing, the aforementioned dummy client testing suite application – as mentioned in part 2.3.2 WebRTC dummy clients – was upgraded to include tests that would help determine the quality of the stream by firstly converting the received packets of a stream into a webm video file format and also by keeping track of the received packets with information of which client received it and at what time with a timestamp in miliseconds.

The webm files created provide a rough idea on the degradation of the video stram as with the increased number of peers the quality can degrade relatively quickly, depending on configurations such as the size of the video being streamed, however, it does not provide quantifiable results that could be interpreted and be useful to understand the whole picture. This is because the degredation of the stream quality is mostly the result of the loss of packets in transit rather than the result of the data in the packets getting corrupted, maening that a better understanding about the quality loss can

be gained from analyzing the packets and identifying the loss depending on the numbers of peers.

Thus, the application was extended to include the packet tracking system that would record the received packets by the receiving peer and the time in a csv file format, which could then wasily be processed further and visualized for either the sum or each individual peer.

4.4.1 Post-processing of the data

The data collected on the packets from the load testing application are processed by a custom made Go script, which looks at each timeframe between a new client was added, which in our setup is configured to be 1 second. Thus, for the period of each one second the script collects the number of all packets received by each client and notes the starting time and end time of the time frame. With this information the script calculates the amount of packets received for each client per second using the following equation:

$$1000 * \frac{\sum packets}{t_{end} - t_{start}}$$

The reason for using the noted start and end timestamps instead of simply summing up the number of packets received is that there could be slight changes in the time which could be enough to interfere with the results. The script also notes and calculates the minimum packets received by a client, as well as the maximum number of packets and the average number of received packets of all the clients per second. The reason behind multiplying by 1000 is because the timestamps are taken in milliseconds and we want to convert it to packets per second.

An illustrative example is provided in the table below, which in two columns presents the data that is being received from the dummy clients. The actual data size or rather the number of timestamps gathered is much grater at any given second or interval, however, for simplicity, the example only has a few entries.

Table 1: Illustrative example of timestamps taken by dummy clients

0	1699335805848	1	1699335806151
1	1699335806060	0	1699335806152
0	1699335806069	1	1699335806162
1	1699335806110	0	1699335806167
0	1699335806116	1	1699335806561
1	1699335806123	1	1699335806573
0	1699335806127	1	1699335806585
1	1699335806136	1	1699335806597
0	1699335806139	New client:	2

As seen in the table above, the number of packets received by peer 0 was 7, while peer 1 received 10 packets. The last timestamp is a notification that a new peer was added and that an average should be calculated for all the existing peers. To do this we take the numbers of packets received as well as the first and last timestamp and insert them into the aforementioned equation.

Peer 0:

$$1000 * \frac{7}{1699335806597 - 1699335805848} = 9.34$$

Peer 1:

$$1000 * \frac{10}{1699335806597 - 1699335805848} = 13.35$$

Using the equation we know that peer 0 and 1 received 9.34 packets and 13.35 packets per second respectively.

4.4.2 Experiment specifications

To further clarify the experimentation environment and characteristics, the experiments were conducted for the video resolutions of 640 by 480 pixels, 1280 by 720 pixels, and 1920 by 1080 pixels, for WebRTC, where:

- Each second a new dummy client was added using the Go testing application,
- The dummy clients increased to 100 unless the connection was cut.

5 RESULTS

After running the tests as described in the methodology, the data compiled was processed for both the server load testing and the client packet stability testing. The main goals of analyzing the data were to present the difference between the presented new model and the traditional model used in production today, and to present the drop

in performance of the new model with the increase in peer users subscribing to a stream. The following subchapters present this two aspects of the research respectively.

Overall the results were as one might anticipate – and as was predicted –, when taking into account the characteristics of the protocols involved and the WebRTC framework's characteristics. It was expected that the proposed model would perform better than the traditional one in terms of cost efficiency by reducing the load on the servers, as it appears to do from the data, and the setup appears to have client side limitations that would be limiting to this technology getting more wide adoption as it appears from the tests and the data that a limit can quite easily be reached – as explained in the second part of the chapter.

5.1 Cost efficiency

Pertaining to the cost efficiency, the main goal of the new model is to decrease the variable cost of the server infrastructure in connection to hasting streaming for the company. The new model is trying to provide this efficiency by essentially trying to bypass the server infrastructure all together whenever possible by creating a P2P connection between the streamers and the subscribers to the streams. Thus, the main goal and indication of the novel model working more efficiently and performing as we would expect would be a lower internet traffic or usage at a certain resolution with a certain number of peers. As explained in the methodology, the experiments were run individually in such a way that the number of peers was increasing by 1 each second until reaching a 100 peers if possible. The data collected was then joined in order to be compared between out HLS benchmark and WebRTC proposed model at different

resolutions of streams. The raw data for resolutions of 640 by 480 pixels, 1280 by 720 pixels, and 1920 by 1080 pixels is available in Appendix 1, Appendix 2, and Appendix 3 respectively. The three figures below (Figure 6, 7, and 8) represent the network speeds of upload and download on the servers, respective to the environment being tested, at resolutions of 640 by 480 pixels, 1280 by 720 pixels, and 1920 by 1080 pixels.

Figure 6: Comparison in upload and download speeds between HLS and WebRTC when increasing the number of peers by one each second until 100 at 640 by 480 pixels

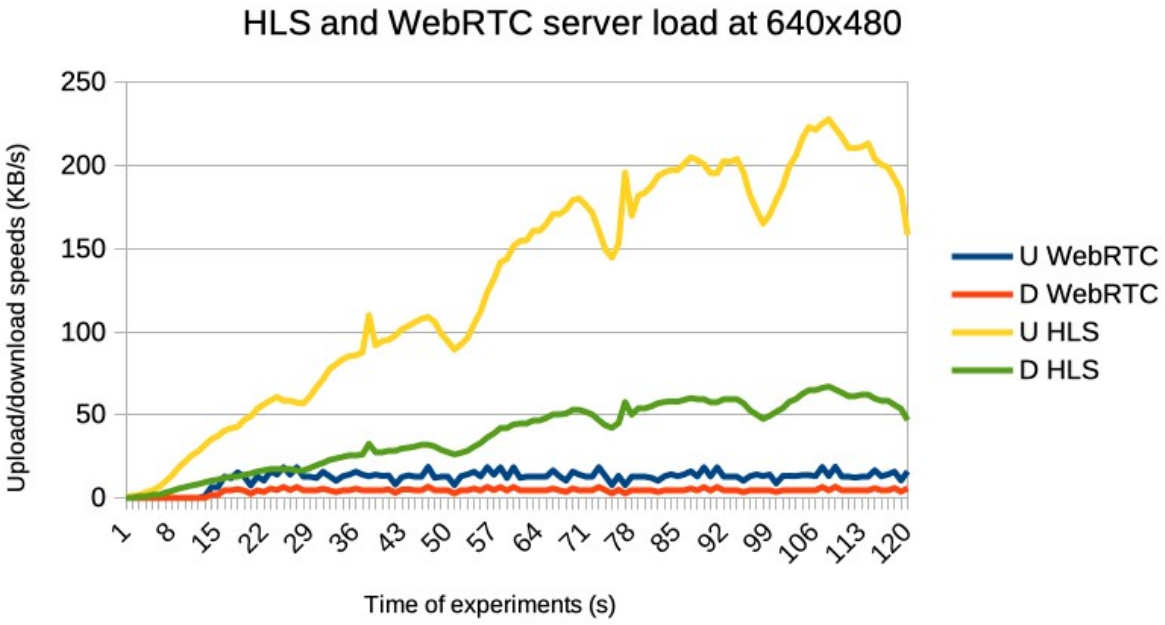


Figure 7: Comparison in upload and download speeds between HLS and WebRTC when increasing the number of peers by one each second until 100 at 1280 by 720 pixels

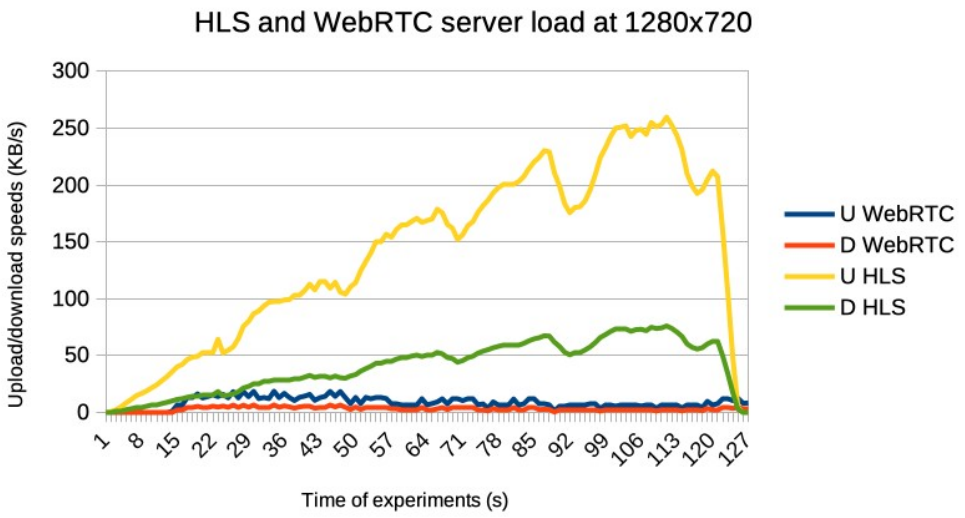
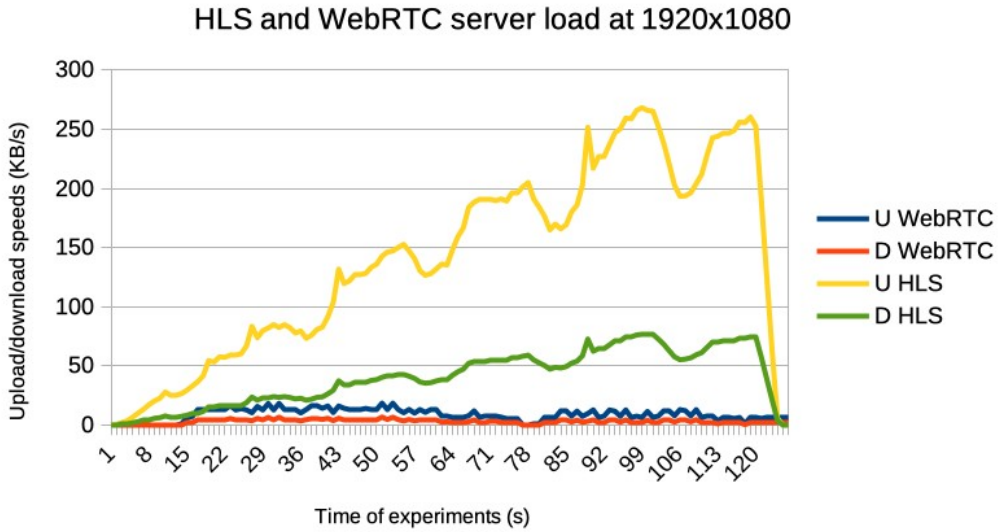


Figure 8: Comparison in upload and download speeds between HLS and WebRTC when increasing the number of peers by one each second until 100 at 1920 by 1080 pixels



From the three figures, it is clearly visible that when it comes to HLS, the cost in performance and the required upload and download speeds increase in a linear fashion, which makes sense as all the traffic must go through the main server, acting as a relay or a proxy. However, when looking at the data for WebRTC, though the upload and download speeds increase somewhat at the beginning, it appears to be more or less a constant. Besides that, both the upload and download speeds in all cases across all examples for WebRTC are constantly below those for HLS, which means that WebRTC does indeed require less bandwidth and thus is more cost efficient.

It should be noted, however, that at resolutions higher than 640 by 480 pixels, the client application actually did not manage to connect all the peers up to a hundred, which explains a drop at resolutions of 1280 by 720 and 1920 by 1080 pixels after the about 60 and 50 second marks respectively. However, we can see the constant nature at 640 by 480 pixels, which does indeed presents near constant upload and download speeds when using WebRTC.

5.2 Client side limitations

As predicted in the technological overview and methodology, there are some significant limitations when it comes to utilizing WebRTC for streaming due to client side performance, as in both higher resolutions selected – 1280 by 720 and 1920 by 1080 pixels – the test ended prematurely due to the streamer client being overburdened by managing all the streams. At the 1280 by 720 pixels resolution, the clients all disconnected at 64 seconds of the experiment, while at the 1920 by 1080 pixels resolution, that occurred even faster at 53 seconds of the experiment. This indicates that

clearly the client can be severely limited, which could lead to large scale streams being a non viable option for streamers more than a few dozen subscribers. The raw data collected from dummy clients and processed into minimum, maximum and average for resolutions 640 by 480 pixels, 1280 by 720 pixels, and 1920 by 1080 pixels is in the Appendix 4, Appendix 5, and Appendix 6 respectively. The below three figures present the same data visually.

Figure 9: The average, minimum and maximum number of packets received per dummy client per second while increasing the number of peers by one each second until 100 at 640 by 480 pixels

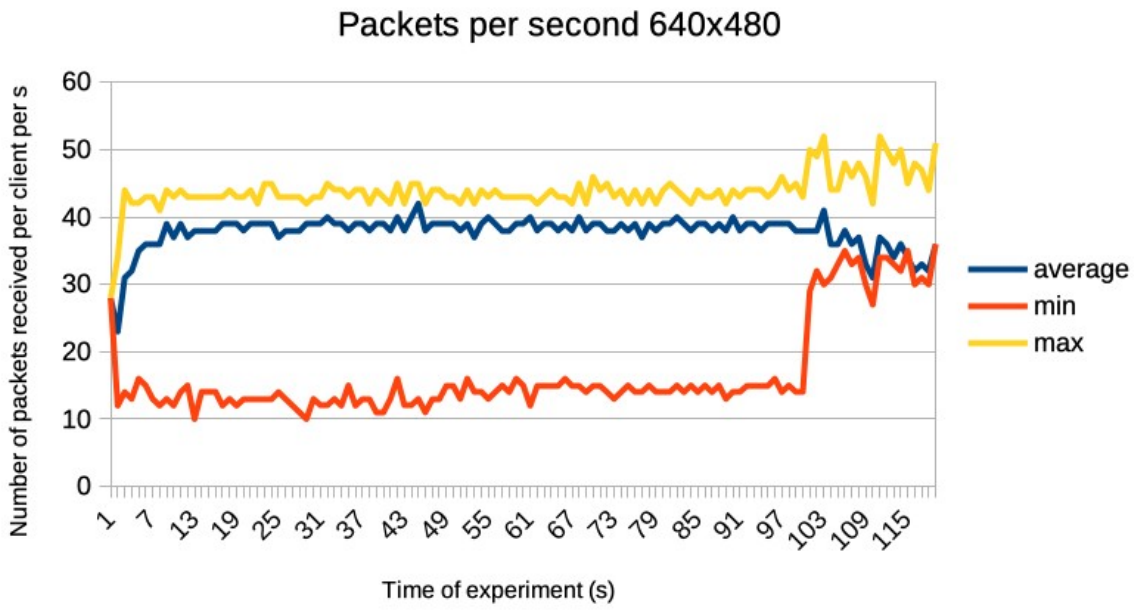


Figure 10: The average, minimum and maximum number of packets received per dummy client per second while increasing the number of peers by one each second until 100 at 1280 by 720 pixels

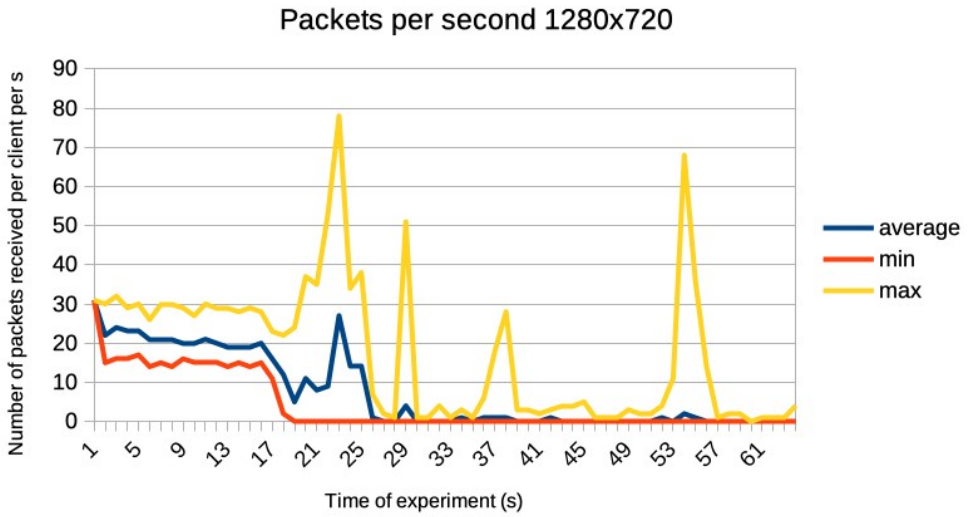
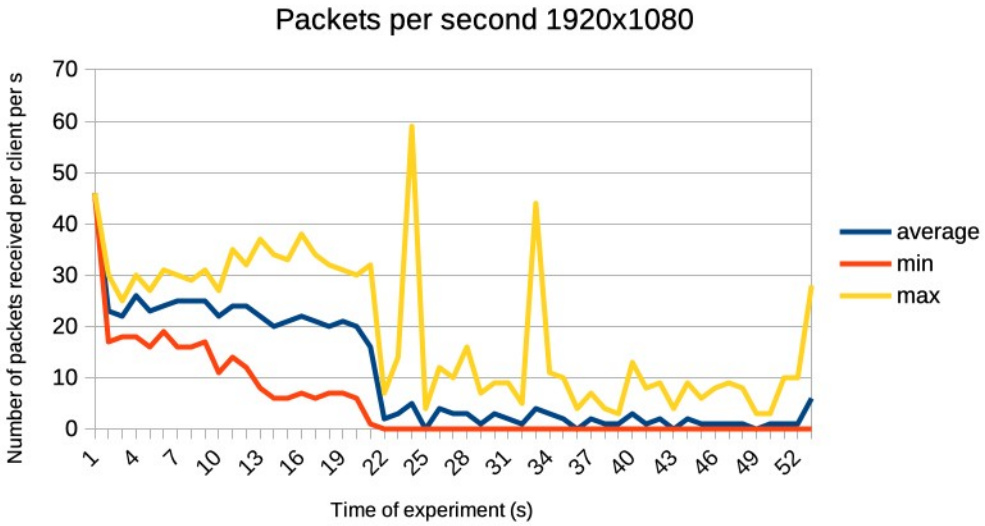


Figure 11: The average, minimum and maximum number of packets received per dummy client per second while increasing the number of peers by one each second until 100 at 1920 by 1080 pixels



From the three figures we can clearly see that the latter two of the higher resolution are very alike one another as the number of packets dropped quite significantly throughout the test, whereas the first appears to have been a lot more stable. This should not be too surprising since the 640 by 480 pixels resolution test was the only one to achieve 100 peers, and while we can clearly see that the minimum number of packets received by a peer was still generally relatively low at about only 15 packets per second, the average number received by users was fairly high at around almost 40 packets per second, indicating that the new connecting peers were likely the ones experiencing initial connection issues, as we see that after all 100 peers managed to connect the minimum number of packets received actually increased slightly indicating increased stability of the streams.

When looking at the higher resolution streams we can see that, although both tests were ended prematurely due to clients being disconnected, the streams were likely no longer consumable in terms of user experience a lot earlier than when the connection was cut by the streamer as, the number of packets received and the stability of packet arrival were compromised at around the 20 peer mark in both cases. When looking at the average number of packets received, we can see a clear dip in performance at the 17 peer mark and 20 peer mark for resolutions of 1280 by 720 pixels, and 1920 by 1080 pixels respectively. It should be noted that although it may appear that the higher resolution had better performance, this was likely due to random chance as the streamer device sometimes performed better and sometimes performed worse likely due to network stability at the time of the experiments. However, when we look at the minimum packets peer user received, we can clearly see that the 1280 by 720 pixels resolution test was already having issues at around 17 peers, while the 1920 by 1080

pixels resolution one at about 10 peers. This means that beyond those thresholds there were already peers who were suffering from jittering and lagging due to not receiving the stream on time. Therefore these limits could be perceived as the maximum available peers for such a P2P setup.

Note that the spikes in performance at higher resolutions after passing the thresholds of what the streamer was able to stably accommodate were most likely due to users receiving packets that were supposed to arrive beforehand but due to performance issues arrived with a delay. Such packets are usually disregarded as they are out of date and cannot be buffered into the stream due to the point of what the user is watching being ahead of what the packets show.

6 DISCUSSION

On the surface, the results of the experiments and this research seem to suggest that WebRTC is overwhelmingly better than the traditional model when it comes to streaming media, as the network load is constant, while the traditional is linear, and the setup is inappropriate for any larger scale streaming with more than a dozen or so peers at higher resolutions like 1920 by 1080 pixels or Full HD, which is what consumers have grown accustomed to; however, the truth is a bit more nuanced than that. We have to take into account a few more things before drawing conclusions such as testing limitations, as well as further development of the proposed model and how it could be improved.

This chapter goes over the two test setups respectively and puts them into perspective regarding real world application and real world expected performance. It also proposes additional research that could be done in a future work.

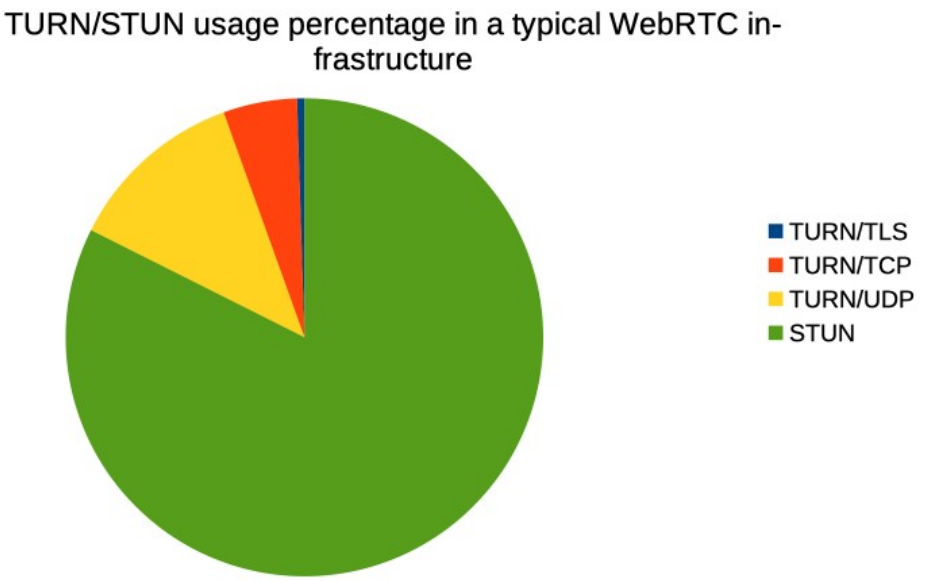
6.1 Real world cost efficiency

Though the results are very optimistic in terms of reducing infrastructure costs of servers, especially with increasing of users of such system, it should be noted that real world results could be less optimal to the test environment. In the test environment we allowed the peers to connect with whichever configuration was most optimal and from results we can assume that most if not all managed to connect using STUN servers only, without having to use a TURN server as a relay. Though this is the default way that WebRTC works and is the most optimal, we can assume that in a real world environment a not so insignificant percentage of users would actually require a TURN server as well which would require more infrastructure and thus we can expect that in larger scale tests we would see that both the benchmark HLS setup and proposed WebRTC setup would increase in bandwidth demand linearly, however, WebRTC would in theory only require bandwidth proportional to the number of peers requiring TURN server configuration and would thus still outperform HLS or any other traditional streaming setup in all cases except in cases where all peers would require a relay, which is fairly untypical.

There is not a lot of data available on the proportion of STUN and TURN used by users of WebRTC applications online, as mostly there are just theories and educated guesses to what the true percentage might be as it is also likely changing over time somewhat,

however, the article by Philipp Hancke (2017) titled “What kind of TURN server is being used?” is, to the best of my knowledge, the only publicly available research with such data. According to Hancke 17,7 percent of calls get relayed in a typical WebRTC infrastructure, so we might assume that the infrastructure cost of the proposed model would be lower by about 80 percent as compared to a benchmark in a real world deployment.

Figure 12: TURN/STUN usage in typical WebRTC infrastructures (%)



Source: Hancke (2017).

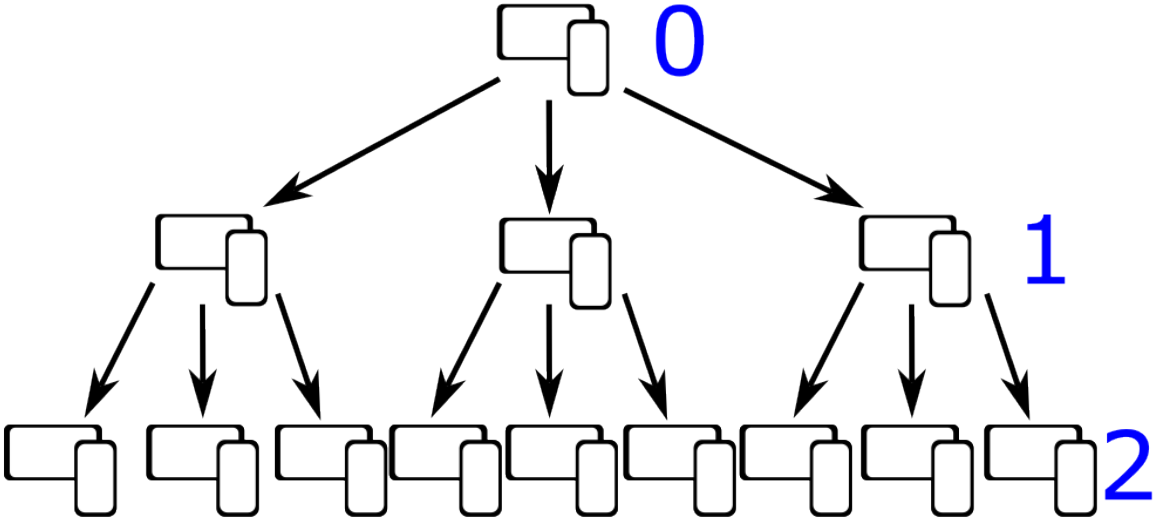
A more problematic trend of inefficiency might be observer if the streamer peers had a higher tendency to not be able to accommodate a direct P2P connection, as that would by default mean that all the peers connecting to their streams would also use a relay and thus the TURN usage percentage could be significantly higher, making this proposed model less appealing. This likely would be a more problematic situation for streamers

who stream from mobile devices connected to a cellular network, which maybe also change towers from time to time or for streamers behind an unusually strict firewall meaning it would be more beneficial for a platform utilizing WebRTC to have streamers stream from their homes as opposed to from on to go for example. This, however is not yet researched and might be worth testing in future research of testing on a live platform with real users for example, as the results would be more accurate and easier to obtain.

6.2 Overcoming limitations

The results of the testing allude to severe limitations of the proposed model. As most modern streaming platforms support for up to thousands of peers connecting to the same stream, one might be led to believe that this model is not suitable for this role, at least when it comes to streams of resolutions higher than 640 by 480 pixels, which is what end users have come to expect in terms of user experience. However, this limitation could be mitigated by introducing a multi-level approach, which would have more than one layer of users or nodes. This would mean that some subscriber peers would also act as secondary streamer and would stream the same stream to others. Such a setup would result in a tree like network structure as shown on the Figure 13 below.

Figure 13: Tree like secondary P2P node network



As seen on the figure, using secondary nodes or users as sort of relays, it would be possible to create a tree like structure with multiple layers, where the layers would have C^L number of peers watching and in some cases re-streaming a stream, where C is the number of connections possible before the network becomes overburdened and L being the number of the layer (represented in blue on Figure 13). Although the figure above only shows 3 connections per peer, in reality we could assume that at 1280 by 720 pixels the number of connections could be 17 and at 1920 by 1080 pixels 10, as was determined from the tests. With this assumption we can calculate the number of peers per layer as shown in the table below.

Table 2: The number of peers per layer for 1280 by 720 and 1920 and 1080 pixels with outgoing connections per peer limited to 17 and 10 respectively

	1280x720	1920x1080
0	1	1
1	17	10
2	289	100
3	4913	1000
4	83521	10000
5	1419857	100000
6	24137569	1000000
7	410338673	10000000
8	6975757441	100000000
9	118587876497	1000000000
10	2015993900449	10000000000

As seen from the table, it would only take 5 layers to reach a million peers at 1280 by 720 pixels, while at 1920 by 1080 it would take 6 layers. This system would of course have to account for quite a few cases such as a peer in the middle disconnecting or losing performance, restructuring the tree when necessary, perhaps taking into account the type of P2P connection possible – whether that is STUN or TURN – when connecting to optimize infrastructure cost saving, perhaps taking into account the geographical distribution of secondary peers, their parent and child nodes, and so on. With this kind of addition to the proposed model it should be possible to reach a number

of peers comparable to what the traditional models in production currently accommodate, without sacrificing performance and user experience, but this would have to be further researched, perhaps in a live production of such a model.

7 CONCLUSIONS

Video streaming, which represents a very sizable part of data transmitted online, has been on the rise in the past few year due to the influx of users and interest in live content that is fresh and relevant to the users. This trend has meant that the business of live streaming has been increasing, but so have the infrastructure costs associated with it. This research has proposed a fairly radical alternative model to the traditional one by taking advantage of the WebRTC P2P framework. Though this framework has become widely adopted by companies, as well as software such as browsers in recent years, especially due to the increase of online meeting through software like Zoom due to the pandemic, this technology has thus far not been implemented into live streaming platforms and other such software due to the complexity of implementation and perceived limitations.

The research conducted in this research confirms both the high potential for significant cost optimization, as well as the high limitations and complexity of this P2P model. Though this research does not definitively prove the possibility of using WebRTC in effective ways to replace the current streaming infrastructure, it does provide some clarity and perspective on the possibilities, that could be explored in further research or projects. The conclusion is that it should be possible to at least in part replace the existing infrastructure with a more cost efficient model, however that would likely

require a lot of work, especially pertaining to building the proposed tree like network infrastructure, as well as an efficient system for determining the type of connection a specific peer is capable of creating, and sorting them appropriately in the network infrastructure.

The main findings of the research are:

1. Though it is not the typical use case, it is certainly feasible to use WebRTC for the purposes of streaming content between users,

2. The proposed model is indeed more cost efficient compared to the traditional setups as was see in the comparison of server loads, though larger scale testing would be required to get precise cost differences,

3. There are significant limitations in terms of scalability, especially at higher resolutions, as at 1920 by 1080 pixels, which is considered as the norm, no more than maybe a dozen subscribers are possible to connect without introducing secondary nodes, which would make the entire network structure a lot more complicated.

The main limitation of the research is that all tests were done as essentially a simulation, which cannot accurately represent the real world conditions, which are a crucial part of WebRTC, as the framework is designed around the very complex and diverse network environments. As such, it can be assumed that the efficiency in a real world application might not be as ideal as in the simulation.

Something that could be further researched in the future is the implementation of the proposed secondary node system to overcome the fairly limited number of peers a single node can accommodate. It would also be of relevance to evaluate and measure how efficient such a network would be if it were implemented as a real product.

REFERENCES

1. Adobe. (2021). *Adobe Flash Player EOL General Information Page*. Retrieved on January 10. 2024 from: https://www.adobe.com/products/flashplayer/end-of-life.html
2. Coturn. (2023). *Coturn*. Retrieved on January 10. 2024 from: https://github.com/coturn/coturn
3. De Cicco, L., and Mascolo, S. (2014). An Adaptive Video Streaming Control System: Modeling, Validation, and Performance Evaluation. *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 526-539. Available on: https://ieeexplore.ieee.org/abstract/document/6493502?casa_token=Rk4Txz8YKu8AAAAA:1uBUmdk0c38SyLakINnm08yM3q33O63O9wRvlXJeJLt7QTViXoObKtNrq43TLzRlL3-uqYcf
4. Durak, K., Akcay, M., N., Erinc, Y., K., Pekel, B., and Begen, A., C. (2020). Evaluating the Performance of Apple's Low-Latency HLS *IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, Tampere, Finland, 2020, pp. 1-6. Available on: https://ieeexplore.ieee.org/abstract/document/9287117
5. Feldmann, A., Gasser, O., Lichtblau, F., Pujol, E., Poese, I., Dietzel, C., Wagner, D., Wichtlhuber, M., Tapiador, J., Vallina-Rodriguez, N., Hohlfeld, O., and Smaragdakis, G. (2021). A Year in Lockdown: How the Waves of COVID-19 Impact Internet Traffic. *Communications of the ACM 64, 7 (July 2021)*.
6. Google for Developers. Real-time communication for the web. *WebRTC*. Retrieved on January 10. 2024 from: https://webrtc.org

7. Hancke., P. (2017). What kind of TURN server is being used? *Medium*. Retrieved on January 10. 2024 from: https://medium.com/@fippo/what-kind-of-turn-server-is-being-used-d67dbfc2ff5d
8. Nginx. (2023). *Nginx*. Retrieved on January 10. 2024 from: https://nginx.org/en/
9. Nunez, L., and Toasa, R., M. (2020). Performance evaluation of RTMP, RTSP and HLS protocols for IPTV in mobile networks. 1*5th Iberian Conference on Information Systems and Technologies (CISTI)*. Seville, Spain, 2020, pp. 1-5. Available on: https://ieeexplore.ieee.org/abstract/document/9140848
10. OBS Project. (2023). *OBS Studio*. Retrieved on January 10. 2024 from: https://obsproject.com
11. Odlyzko, A. M. (2003). Internet traffic growth: sources and implications. *Optical Transmission Systems and Equipment for WDM Networking II.* Available on: https://doi.org/10.1117/12.512942
12. Raboof. (2023). *Nethogs*. Retrieved on January 10. 2024 from: https://github.com/raboof/nethogs
13. Ramos-Chavez., R., Karagkioules., T., and Mekuria., R. (2020). A scalable load generation framework for evaluation of video streaming workflows in the cloud. *Association for Computing Machinery, New York, NY, USA, 255–260*. Available on: https://doi.org/10.1145/3339825.3394930
14. Sanni, M., L., Hashim, A., A., Anwar, F., Ahmed, G., S., M., and Ali, S. (2013). How to model wireless mesh networks topology. *IOP Conference Series: Materials Science and Engineering, Volume 53, 5th International Conference on Mechatronics (ICOM'13) 2–4 July 2013, Kuala Lumpur, Malaysia.* Available on: https://iopscience.iop.org/article/10.1088/1757-899X/53/1/012037/meta

15. Stream. *TURN Server*. Retrieved on January 10. 2024 from: https://getstream.io/glossary/turn-server/
16. Vegibit. *What Is A STUN Server And How Does It Work*. Retrieved on January 10. 2024 from: https://vegibit.com/what-is-a-stun-server-and-how-does-it-work/
17. Vučić., D., Skorin-Kapov., L., & Sužnjević., M. (2016). The impact of bandwidth limitations and video resolution size on QoE for WebRTC-based mobile multi-party video conferencing. *screen, 18*, 19.
18. Wilbert., M. (2023). What Is HLS Streaming and When Should You Use It [2023 Update]. *Dacast*. Retrieved on January 10. 2024 from: https://www.dacast.com/blog/hls-streaming-protocol/

APPENDICES

Appendix 1: Upload and download speeds for HLS and WebRTC with clients increasing by one each second until 100 at resolution of 640 by 480 pixels

U WebRTC	D WebRTC	U HLS	D HLS				
0.03398	0.03867	0.471484	0.196094	15.804	5.6416	85.7707	25.5029
0.03398	0.03867	1.34395	0.473437	14.084	4.7719	87.6711	26.0957
0.03398	0.03867	1.83184	0.702539	13.072	4.6082	109.964	32.6086
0.06210	0.06210	3.39824	1.26777	14.038	4.7332	91.8781	27.1705
0.08789	0.11055	4.49629	1.62637	13.288	4.6494	94.3455	27.7563
0.08789	0.11055	6.65566	2.34238	13.51	5.1732	95.1953	27.9879
0.08789	0.11055	9.81563	3.25137	8.1633	3.1643	97.6605	28.7344
0.05390	0.06015	13.5189	4.2707	12.579	5.0658	101.554	29.7398
0.05390	0.06015	18.2453	5.61582	13.614	5.2037	103.502	30.2689
0.05390	0.06015	21.8963	6.5457	13.212	4.7605	105.773	30.9729
0.05390	0.06015	25.593	7.53672	13.078	4.659	107.915	31.6768
0.05390	0.06015	28.0939	8.28555	18.72	6.7707	108.969	32.0057
0.98867	0.15391	31.6699	9.4084	12.192	4.5838	105.887	31.0146
6.4281	2.1768	35.4104	10.4172	13.149	4.4934	98.8391	28.9135
6.4426	2.192	37.3314	11.0646	13.166	4.6094	94.2543	27.5613
12.855	4.2676	40.468	12.0145	7.8209	2.7381	89.3008	26.2227
11.904	4.2637	41.9363	12.4754	13.102	4.6488	92.4205	27.1984
15.406	5.3035	42.8336	12.7785	14.249	4.891	96.2973	28.4123
12.875	4.4865	47.0113	14.0365	15.779	5.6635	104.672	30.9258
7.5912	2.5873	49.0678	14.648	12.998	4.4881	112.357	33.1203
12.92	4.5711	53.6811	15.9471	18.506	6.6467	123.893	36.4496
10.533	3.6697	56.373	16.6502	14.062	4.785	131.709	38.8031
16.01	5.6582	58.6773	17.2311	18.411	6.5545	141.881	41.6686
14.139	4.8016	60.7754	17.8047	12.057	4.5074	143.958	42.2389
18.45	6.6303	58.7549	17.2262	18.492	6.6547	151.648	44.2975
14.074	4.7715	58.2527	17.1271	12.125	4.466	154.712	45.1486
18.465	6.615	57.4551	16.9791	13.123	4.5439	155.128	45.225
12.811	4.4361	56.8947	16.8543	13.126	4.6061	161.282	46.9648
12.921	4.4826	61.048	18.0125	13.131	4.4969	161.015	47.0785
12.018	4.4041	66.5951	19.6865	13.172	4.6887	164.891	48.2109
15.745	5.559	71.4008	21.1514	16.607	5.6824	170.964	50.1012
13.048	4.5578	77.6734	22.916	13.06	4.5635	171.118	50.0473
10.316	3.5492	80.4512	23.7963	10.466	3.684	173.672	50.7893
13.077	4.5725	83.4461	24.726	15.87	5.7324	179.228	52.6484
14.107	4.8045	85.3414	25.3172	14.094	4.7967	180.235	52.8695
13.167	4.6699	176.359	51.8486	13.101	4.6074	182.136	52.7537

12.982	4.5053	171.646	50.2109	14.219	4.8635	173.406	50.2242
18.517	6.5926	161.045	46.8426	13.176	4.8338	165.144	47.6133
13.207	4.7051	149.488	43.6822	14.093	4.8244	170.4	49.408
7.823	2.7674	144.669	42.2543	8.8102	3.651	179.394	51.877
13.304	4.9273	153.162	45.1018	13.268	4.8635	187.629	54.0564
7.751	2.7803	195.823	57.6738	13.341	4.8897	199.766	57.9031
13.182	4.7838	169.876	50.1268	13.198	4.833	206.183	59.5701
13.227	4.8303	181.909	53.65	13.338	4.9533	216.733	62.7266
13.072	4.5305	183.78	54.0814	13.34	4.6383	223.332	65.2291
12.139	4.5379	187.761	55.2107	13.187	4.5361	221.538	64.7311
10.39	3.6002	193.755	57.0424	18.6	6.6709	225.225	66.1844
13.089	4.6687	195.994	57.7818	13.241	4.5197	227.931	67.0676
14.282	4.9693	197.607	58.1982	18.803	6.8039	222.56	65.2457
13.108	4.6258	196.991	57.9635	13.067	4.8018	217.332	63.4197
14.232	4.9281	201.125	59.0697	13.184	4.7502	210.782	61.6154
16.056	5.7328	205.015	60.1289	12.251	4.6404	210.559	61.6484
13.011	4.5354	203.319	59.8096	13.124	4.6023	211.211	61.8607
18.444	6.5809	200.695	59.2529	13.096	4.6977	213.344	62.3248
12.994	4.4471	195.242	57.7258	16.688	5.8537	204.199	59.8277
18.453	6.624	195.181	57.5641	13.06	4.7498	200.621	58.7801
13.092	4.6963	202.744	59.5574	14.211	5.0561	199.042	58.2967
13.017	4.4914	202.202	59.2086	15.808	5.9123	192.29	56.0459
13.065	4.6	204.036	59.573	10.587	3.8635	184.775	53.6682
10.37	3.5389	196.245	57.0943	16.052	5.8621	158.345	46.773

Appendix 2: Upload and download speeds for HLS and WebRTC with clients increasing by one each second until 100 at resolution of 1280 by 720 pixels

U WebRTC	D WebRTC	U HLS	D HLS				
0.044531	0.0375	0.24043	0.076367	13.225	4.948	99.299	28.7398
0.044531	0.049219	0.879687	0.259766	10.539	3.8478	102.761	29.7437
0.044531	0.049219	2.41309	0.718164	13.323	4.9666	102.703	29.7062
0.044531	0.049219	5.01035	1.46699	14.38	5.0258	107.177	31.051
0.087891	0.097656	8.41973	2.44512	15.962	5.8369	112.744	32.6156
0.087891	0.097656	11.4174	3.31621	10.703	3.7688	107.805	30.85
0.087891	0.097656	14.8139	4.29199	13.085	4.6303	114.752	32.0672
0.087891	0.097656	16.7188	4.84219	14.146	4.9064	115.081	32.0336
0.087891	0.097656	18.9324	5.47109	18.465	6.5695	109.275	30.457
0.087891	0.097656	21.7291	6.17656	14.206	4.9221	114.439	31.9877
0.087891	0.097656	24.2998	6.88438	18.407	6.5309	105.859	30.5539
0.087891	0.097656	27.9633	7.88496	13.089	4.5367	104.053	30.1385
0.043359	0.048438	31.5303	8.94766	7.7869	2.6674	110.501	32.1051
0.98867	0.15391	35.751	10.1703	13.209	4.7945	114.143	33.3221
6.4789	2.151	40.225	11.5176	7.8756	2.7768	124.705	36.3506
6.4934	2.2049	42.3004	12.1652	13.285	4.8141	132.786	38.5645
12.822	4.1936	46.5621	13.4008	12.087	4.5049	140.444	40.6971
12.846	4.3658	48.693	14.0121	13.328	4.8496	149.987	43.2727
16.325	5.4061	49.3611	14.1416	13.187	4.6895	150.343	43.3773
12.802	4.4701	52.1465	14.9387	12.275	4.5773	156.73	45.227
13.978	4.6996	52.374	15.0279	7.8943	2.785	154.174	44.7518
15.517	5.5352	52.8146	15.0432	7.9568	2.8197	160.942	46.9164
13.999	4.7518	64.376	18.39	6.7049	2.3695	164.881	48.4197
15.864	5.6541	52.3008	14.6811	6.7283	2.4281	165.146	48.5527
12.974	4.5488	54.7609	15.2951	6.5764	2.2807	168.046	49.608
18.27	6.4986	57.7441	16.299	6.7893	2.4441	170.832	50.4668
12.887	4.4539	64.9898	18.3213	12.105	4.3994	167.21	49.2617
18.305	6.4482	75.7971	21.6	6.7412	2.4793	168.756	49.9188
13.944	4.6521	80.0992	22.9141	7.8494	2.7287	170.207	50.3209
18.544	6.5814	86.9154	24.8937	9.2645	3.2834	178.778	52.7938
12.254	4.5525	89.3123	25.6377	11.99	4.3439	175.591	51.5834
13.164	4.5668	93.5748	26.9906	7.6164	2.6277	165.481	48.3631
12.239	4.5172	96.998	27.9945	11.893	4.4006	162.105	47.3664
18.64	6.6512	97.3865	28.1039	11.966	4.4889	152.152	44.2234
13.046	4.6375	97.8279	28.1922	10.494	3.8752	156.418	45.6486
16.795	5.8074	98.9623	28.518	12.11	4.5021	164.323	48.2018

12.171	4.6807	167.827	49.4686	5.5508	2.1941	250.118	73.4578
6.8955	2.5051	176.046	52.1609	6.4855	2.3008	250.885	73.6332
7.5574	2.4779	181.897	53.9242	6.8019	2.5412	252.056	73.9365
3.9512	1.4467	186.767	55.2887	6.7767	2.4313	242.665	71.3754
9.308	3.4676	193.194	56.9188	6.6545	2.3691	248.017	72.91
6.6127	2.3937	197.646	58.1191	5.667	2.1828	249.065	73.1689
6.5678	2.3289	200.401	58.8291	6.3818	2.1451	244.54	71.9457
6.7455	2.5322	201.056	59.0295	6.5795	2.3098	255.046	74.9049
11.848	4.3613	200.281	58.7271	3.7148	1.3748	251.181	73.8463
6.5461	2.4018	203.058	59.6148	6.6348	2.4869	253.351	74.3803
7.6367	2.6113	207.497	60.7918	6.6816	2.5768	259.751	76.1137
11.886	4.3736	214.439	63.0068	6.6262	2.5119	252.85	73.9943
12.005	4.5107	220.376	64.7057	6.691	2.5822	243.678	70.7914
7.5508	2.5721	224.362	65.7121	3.9764	1.4363	231.168	66.9592
7.8096	2.7832	230.284	67.4826	6.5924	2.4231	210.145	60.3754
6.7287	2.5678	229.159	67.2035	6.6338	2.4219	199.279	57.324
1.3521	0.48262	210.841	61.8309	6.4461	2.2854	192.629	55.7375
5.7553	2.2986	199.305	58.0605	4.3887	1.6178	195.881	56.9721
5.6199	2.1621	183.763	53.0617	9.7894	3.4869	204.737	60.2227
6.5266	2.243	175.679	50.6766	6.6086	2.2428	212.312	62.5131
6.7477	2.5076	180.289	52.1182	7.757	2.5559	207.366	62.6031
6.5543	2.4502	180.914	52.8881	12.089	4.2732	159.971	49.799
6.5461	2.3951	186.035	54.9467	12.272	4.2984	105.7	34.3023
7.6852	2.7098	195.536	57.6402	10.48	3.6512	47.6117	17.1609
7.6852	2.7227	208.937	61.4537	12.519	4.5408	3.96484	2.81797
3.4873	1.2605	224.104	65.9502	8.1568	2.7805	0.237109	0.430273
6.6469	2.5467	232.743	68.4447	8.3545	2.9709	0.208984	0.015234
6.7492	2.4744	242.554	71.2512				

Appendix 3: Upload and download speeds for HLS and WebRTC with clients increasing by one each second until 100 at resolution of 1920 by 1080 pixels

U WebRTC	D WebRTC	U HLS	D HLS				
0.033984	0.025781	0.027343	0.026953	13.151	4.6906	73.3861	21.0596
0.051562	0.025781	0.766602	0.256055	16.672	5.7762	76.0482	21.9354
0.077344	0.074219	2.05215	0.635742	15.914	5.8287	80.8646	23.4916
0.077344	0.074219	3.80801	1.16816	14.27	4.8666	82.774	23.9158
0.053906	0.060156	6.82324	2.0123	16.165	5.7998	91.5691	26.4248
0.053906	0.071875	10.2111	2.95996	10.705	3.8129	103.647	29.4961
0.053906	0.071875	13.4311	3.89219	16.172	5.8299	131.744	37.4305
0.053906	0.071875	17.2959	4.89043	14.246	4.935	119.606	33.9154
0.031641	0.049219	20.4928	5.75918	13.199	4.599	121.986	34.5301
0.059766	0.051562	22.6992	6.33184	13.027	4.5043	127.009	36.0148
0.096094	0.11289	27.8955	7.75547	13.111	4.6533	127.511	36.2287
0.096094	0.11289	25.074	6.93027	14.106	4.8096	128.298	36.3477
0.1	0.11523	25.1211	7.05488	13.079	4.6305	133.359	37.7914
1.1033	0.31367	26.576	7.51094	13.079	4.6762	136.05	38.4814
6.4949	2.3279	29.5848	8.35156	18.563	6.8131	142.802	40.4482
7.3652	2.3475	33.0672	9.44141	13.147	4.7697	146.319	41.4521
12.844	4.4359	36.7471	10.3572	18.594	6.657	147.182	41.5383
12.772	4.3289	41.7143	11.6105	13.152	4.6809	150.222	42.408
12.945	4.4773	54.4715	15.235	10.497	3.6045	152.72	43.0369
12.966	4.5031	53.3666	14.9705	13.263	4.7619	147.161	41.3734
12.975	4.4457	57.6217	16.2201	10.218	3.6801	140.82	39.5055
12.992	4.5488	57.3357	15.9998	13.16	4.6404	130.458	36.4793
16.483	5.5934	59.0809	16.4736	10.83	3.8924	126.565	35.5033
12.87	4.4656	59.3516	16.4889	13.323	4.702	128.001	35.9217
14.015	4.7553	60.3205	16.8098	13.354	4.7332	131.875	37.3559
12.858	4.5658	67.075	19.0836	7.7875	2.5846	136.068	38.5811
10.509	3.7023	83.4584	23.7449	8.0393	2.8527	135.415	38.3988
15.81	5.658	73.9002	21.1139	6.6682	2.3143	148.088	42.1254
12.82	4.474	79.8928	22.7516	6.7068	2.4209	159.209	45.0984
18.268	6.4932	82.0326	23.3559	6.7213	2.4361	166.649	47.2969
12.896	4.4465	84.8605	24.1096	7.9115	2.8465	184.107	52.2582
18.375	6.6379	82.507	23.4547	12.043	4.4354	188.426	53.3619
13.067	4.7152	84.9002	24.1037	6.732	2.5471	190.625	54.1309
12.68	4.5223	82.4021	23.3805	7.6668	2.6408	190.52	54.1934
13.035	4.7676	77.9174	22.1707	7.9664	2.9242	191.189	54.6656
10.345	3.6367	79.5014	22.7279	7.9664	2.9242	189.705	54.7371

6.7754	2.527	191.132	55.2076	11.546	4.1037	265.92	76.5195
5.8582	2.4484	189.399	54.8676	6.6631	2.3809	265.287	76.2693
5.7191	2.3193	195.96	56.6195	7.5979	2.4746	252.646	72.7834
5.7191	2.3193	196.798	56.9205	12.023	4.2676	238.263	68.4051
0.42129	0.2668	201.393	58.2221	12.036	4.3063	220.936	62.9666
0.30312	0.23867	205.036	59.1203	7.999	2.632	202.767	57.676
1.1594	0.32813	191.314	55.1699	13.124	4.5002	193.524	55.2502
1.3371	0.47988	184.295	52.6527	12.213	4.5455	193.752	55.8326
6.2916	2.1359	176.508	50.4268	7.8852	2.843	196.335	56.8848
6.4967	2.2953	165.062	47.3252	13.091	4.6359	204.067	59.3937
6.7072	2.4643	169.908	48.8654	6.5584	2.4402	211.937	61.2365
11.778	4.1906	165.888	48.2986	7.7174	2.7588	228.366	65.718
11.778	4.2023	169.358	49.4398	7.7695	2.674	242.854	69.8656
7.6531	2.627	180.265	52.3828	3.7012	1.2031	244.18	70.0459
11.864	4.2373	186.025	54.0344	6.6219	2.3414	246.696	71.0342
7.4486	2.4066	202.841	58.5865	6.7406	2.4473	246.399	70.9385
9.8014	3.2877	251.63	72.6951	5.5848	2.1135	248.994	71.6189
12.876	4.4447	217.107	62.4936	6.5094	2.26	256.082	73.8727
6.5391	2.3236	227.219	65.1373	1.3746	0.46582	255.633	73.7049
7.6516	2.5691	227.196	65.0857	6.3705	2.1578	260.359	74.8326
12.842	4.4207	237.234	67.7965	6.5777	2.317	252.578	74.3635
11.775	4.2088	246.934	70.7686	5.6477	2.2049	191.425	57.7398
7.5414	2.516	250.531	71.7596	6.793	2.4945	126.896	39.542
12.864	4.4566	259.452	74.4141	7.0039	2.4584	63.2771	21.3471
6.5477	2.3283	258.793	74.3029	6.6059	2.4896	5.61777	3.7541
7.6152	2.5402	265.954	76.2037	6.6105	2.4639	0.440625	0.570703
6.6086	2.3174	268.21	77.098	6.7117	2.5525	0.213086	0.036328

Appendix 4: Number of packets received per client per second in average, minimum, and maximum while increasing peers by 1 per second till 100 at resolution of 640 by 480 pixels

Average	Minimum	Maximum	38	11	43
28	28	28	39	10	42
23	12	34	39	13	43
31	14	44	39	12	43
32	13	42	40	12	45
35	16	42	39	13	44
36	15	43	39	12	44
36	13	43	38	15	43
36	12	41	39	12	44
39	13	44	39	13	44
37	12	43	38	13	42
39	14	44	39	11	44
37	15	43	39	11	43
38	10	43	38	13	42
38	14	43	40	16	45
38	14	43	38	12	42
38	14	43	40	12	45
39	12	43	42	13	45
39	13	44	38	11	42
39	12	43	39	13	44
38	13	43	39	13	44
39	13	44	39	15	43
39	13	42	39	15	43
39	13	45	38	13	42
39	13	45	39	16	44
37	14	43	37	14	42
38	13	43	39	14	44
38	12	43	40	13	43
39	14	44	38	14	43
38	15	43	39	15	44
38	14	43	38	13	42
39	16	43	40	14	44
39	15	43	38	14	43
40	12	43	39	15	44
38	15	42	39	15	44
39	15	43	38	15	44
39	15	44	39	15	43

38	15	43	39	16	44
39	16	43	39	14	46
38	15	42	39	15	44
40	15	45	38	14	45
38	14	42	38	14	43
39	15	46	38	29	50
39	15	44	38	32	49
38	14	45	41	30	52
38	13	43	36	31	44
39	14	44	36	33	44
38	15	42	38	35	48
39	14	44	36	33	46
37	14	42	37	34	48
39	15	44	33	30	46
38	14	42	31	27	42
39	14	44	37	34	52
39	14	45	36	34	50
40	15	44	34	33	48
39	14	43	36	32	50
38	15	42	34	35	45
39	14	44	32	30	48
39	15	43	33	31	47
32	30	44	36	36	51

Appendix 5: Number of packets received per client per second in average, minimum, and maximum while increasing peers by 1 per second till 100 at resolution of 1280 by 720 pixels

Average	Minimum	Maximum	0	0	1
31	31	31	1	0	3
22	15	30	0	0	1
24	16	32	1	0	6
23	16	29	1	0	18
23	17	30	1	0	28
21	14	26	0	0	3
21	15	30	0	0	3
21	14	30	0	0	2
20	16	29	1	0	3
20	15	27	0	0	4
21	15	30	0	0	4
20	15	29	0	0	5
19	14	29	0	0	1
19	15	28	0	0	1
19	14	29	0	0	1
20	15	28	0	0	3
16	11	23	0	0	2
12	2	22	0	0	2
5	0	24	1	0	4
11	0	37	0	0	11
8	0	35	2	0	68
9	0	53	1	0	36
27	0	78	0	0	14
14	0	34	0	0	1
14	0	38	0	0	2
1	0	7	0	0	2
0	0	2	0	0	0
0	0	1	0	0	1
4	0	51	0	0	1
0	0	1	0	0	1
0	0	1	0	0	4
0	0	4			

Appendix 6: Number of packets received per client per second in average, minimum, and maximum while increasing peers by 1 per second till 100 at resolution of 1920 by 1080 pixels

Average	Minimum	Maximum	3	0	10
46	46	46	3	0	16
23	17	30	1	0	7
22	18	25	3	0	9
26	18	30	2	0	9
23	16	27	1	0	5
24	19	31	4	0	44
25	16	30	3	0	11
25	16	29	2	0	10
25	17	31	0	0	4
22	11	27	2	0	7
24	14	35	1	0	4
24	12	32	1	0	3
22	8	37	3	0	13
20	6	34	1	0	8
21	6	33	2	0	9
22	7	38	0	0	4
21	6	34	2	0	9
20	7	32	1	0	6
21	7	31	1	0	8
20	6	30	1	0	9
16	1	32	1	0	8
2	0	7	0	0	3
3	0	14	1	0	3
5	0	59	1	0	10
0	0	4	1	0	10
4	0	12	6	0	28