National College of Ireland

# Scaling WebRTC video broadcasting using partial mesh model with location based signalling

MSc Research Project
Cloud Computing


Adesh Rohan D'Silva
Student ID: x18176097

School of Computing
National College of Ireland


Supervisor: Manuel Tova-Izquierdo

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

National College of Ireland

| | |
|---|---|
| **Student Name:** | Adesh Rohan D'Silva |
| **Student ID:** | x18176097 |
| **Programme:** | Cloud Computing |
| **Year:** | 2020 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Manuel Tova-Izquierdo |
| **Submission Due Date:** | 17/08/2020 |
| **Project Title:** | Scaling WebRTC video broadcasting using partial mesh model with location based signalling |
| **Word Count:** | 6531 |
| **Page Count:** | 21 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

| | |
|---|---|
| **Signature:** | |
| **Date:** | 16th August 2020 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies). | ☐ |
| **Attach a Moodle submission receipt of the online project submission**, to each project (including multiple copies). | ☐ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | ☐ |

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Scaling WebRTC video broadcasting using partial mesh model with location based signalling

Adesh Rohan D'Silva
x18176097

**Abstract**

The development and release of the new WebRTC standards and the deprecation of flash technology lead to the rapid adoption of WebRTC technology for developing various media based applications like video conferencing, broadcasting, collaboration, etc. WebRTC does not document any standard or recommended way of creating the signalling network so there have been many attempts at discovering different signalling mechanisms for providing an optimal experience to the end-user. The existing solutions heavily rely on media servers to distribute the media streams defeating the advantages of peer-to-peer networks provided by WebRTC. The main motivation of this research was to optimize the peer-to-peer network to create a scalable architecture for broadcasting video to multiple users. A simple mesh based network is not scalable so this paper suggests using a modified version of mesh network which is scalable and reliable. It makes use of a partial mesh model with redundant connections where all peers are connected based on the nearest location of the peers. The results show that this modified network consumes 80% less bandwidth for broadcaster when compared to the standard broadcaster model. It was also found that this model does not provide infinite scalability and after a certain limit of users a hybrid approach of combining this model with a server model discussed later in the paper would be suitable.

# 1 Introduction

## 1.1 Background

Google first released WebRTC as an attempt to create a standard way of establishing real time communication between browsers. The technology provides browsers a way to create peer-to-peer connections to share audio, video and data streams between the users. It makes use of the Interactive Connectivity Establishment (ICE) framework to even overcome the network complexities with the help of Session Traversal Utilities for NAT (STUN) or Traversal Using Relay around NAT (TURN) server (Dutton; 2013). This gave rise to many innovative video conferencing and collaboration solutions. Google Hangouts (Carlucci et al.; 2016), Discord (Vass; 2018), Zoom (Blum et al.; 2020) and many big companies now use WebRTC in some or the other form to provide real time communication for its users. WebRTC enables peer-to-peer communication but it still needs a signalling server to exchange metadata about the peers for initial connection establishment. As defined in the JavaScript Session Establishment Protocol (JSEP), the signalling is not defined but is instead left to the application in order to provide maximum

compatibility (Uberti and Jennings; 2013). A lot of the research which has been done focuses on optimizing the transfer of streams and scalability of media server responsible for exchanging the streams in a network. The server acts like a Selective Forwarding Unit (SFU) which is responsible for media exchange between all the users in the network. The application therefore provides security, scalability and reliability but at the cost of an expensive service because of the centralized media servers.

## 1.2 Motivation

The video conferencing and broadcasting solutions that exist today do not take advantage of the bandwidth and processing capabilities of the end users. Edan et al. (2018) shows that hybrid networks are possible and can improve the performance of the media exchange. This paper focuses on improving the performance at the user end (i.e. the client side) instead of the server side using similar principles. It proposes an optimum network model by introducing an alternative to the traditional mesh topology thereby taking full advantage of WebRTC technology to create a scalable peer-to-peer network. The best part of this research proposal is that it has potential to be used together with all the existing extensively researched and optimized media gateways/servers to provide infinite scalability to users at a lower cost.

## 1.3 Research question

The research question tries to answer whether a peer-to-peer model for video broadcasting can be made scalable and reliable using a modified mesh network and geolocation optimization?

### 1.3.1 Objectives

To explore the research question, following objectives were decided:

- Design a modified mesh model that scales better than the classical mesh model.
- Evaluate ways to increase reliability of the peer-to-peer network.
- Find if nodes can be optimally connected using geolocation.

### 1.3.2 Methods to meet the objectives

To meet the objectives, following tasks will be performed:

- Research on different network models used in WebRTC signalling.
- Write a WebRTC application using a partial mesh model.
- Make the partial mesh model flexible so that a node can be connected to N other nodes to increase reliability.
- Implement algorithm to calculate closest nodes based on location.
- Use statistics API provided by Chrome browser to verify if the objectives are met.

## 1.4 Limitations

The partial mesh network which has been proposed is designed and tested to work only with video broadcasting and not for any other types of collaboration use case. The primary focus of the paper was to locally optimize the network for users residing close to each other using only peer-to-peer connections. So, it is not recommended to use this model to connect users across the globe. This is why a hybrid architecture is proposed further down this paper to solve this problem. Since, the partial mesh model is still a peer-to-peer based model, without using a hybrid architecture the size of the network cannot be scaled infinitely. Every node in the network establishes N (configurable) connections with other nodes so there is redundancy in the network but this is done to increase reliability of the network.

The paper is structured into following sections: Section 2 discusses the existing literature and research that have contributed towards the optimization of webrtc applications and how these proposals compare with each other and the proposed solution. Section 3 describes the methodology and approach for executing and testing the proposed solution. Section 4 describes the internal working and algorithms used as part of the design specification. Section 5 gives details on the different technologies and frameworks used for implementing the project and libraries implemented for testing and validating the results. Section 6 discusses the results by running the model under different loads and verifying the objectives proposed in the Section 1. The paper is concluded by discussing the final outputs and objectives and giving possibles future lines of work in the last section.

# 2 Related Work

Even before WebRTC was released, there has been significant contributions to the optimization and performance of video conferencing and broadcasting applications. With remote collaboration becoming more and more popular after the introduction of WebRTC, there has been a lot of focus on increasing the scalability of these systems. The three main areas of research for improving the quality and performance of collaboration in WebRTC applications are:

- **Network models**
  There are many different models researched like the centralized P2P network (Ng et al.; 2014) compared with a pure mesh model. There has also been some research on hybrid network models using mesh and star topology to improve performance of the network (Edan et al.; 2018).

- **Distributed architecture**
  With the rise in popularity of microservices and scalable cloud architecture, the research has also shifted from using vertically scalable Multipoint Control Unit (MCU) to a micro MCU architecture (Rodríguez et al.; 2016). There has also been research on creating scalable relay servers using surrogate peers on the cloud (Wu et al.; 2013)

- **Bandwidth optimization**
  The reason it is so difficult to optimize video collaboration applications is because media consumes lot of bandwidth and processing power. So, there has been lot of research on optimizing the bandwidth consumption by selectively sending parts of streams or researching optimized compression techniques (Jansen et al.; 2018).

We will look at these existing researches and how they have tried to solve different issues in WebRTC applications and how our proposal compares to their solutions.

## 2.1 Network Models

The development of WebRTC framework has given native capabilities to the browser to capture audio and video streams from the users and establish peer-to-peer connections with other users. This technology has helped in the development of video collaboration applications which does not require the user to install any plugins and has made it possible to exchange high quality video streams. There have been many signalling solutions proposed for different usecases, the simplest and the most inexpensive being the mesh network (Edan et al.; 2019). The mesh model is highly reliable as every node is connected to every other node so there are no single point of failures in the networks. This model however cannot be used in enterprise applications due to it's inherent limitation on its ability to scale. Edan et al. (2018) has shown in his paper that the media streams puts a lot of stress on the resources of every peer in the network. The encoding of the streams require CPU and transfer of multiple high quality streams consume bandwidth. This paper tries to solve these problems by proposing a scalable mesh model with reliability by creating a partial mesh network.

An alternative to mesh based network is the tree topology which provides a highly scalable network and is suited for a video broadcasting use case. However, as the network grows in size the number of hops between each node increases which might not be acceptable in a real-time use case (El Hamzaoui et al.; 2018). This model also has lower reliability since if one node goes down, all the nodes that were relying on it for receiving the stream will also go down. Our approach is a variation of this model with different optimizations to overcome these issues.

The network formed by interconnected peers is never homogeneous and the conditions of the network keep changing. El Hamzaoui et al. (2018) proposes a dynamic clustered solution which is flexible enough to adapt to different capabilities of the user. A peer-to-peer connection is always reliant on the capabilities of the client so a framework that can adjust based on this can provide better performance. They have proposed a decentralized model which is based on the concept of super node. Clusters are created based on different parameters like available bandwidth of the peers or latency in the network. A super node is then selected from these peers in the cluster based on these capabilities. This ensures higher reliability and provides a scalable solution however the implementation itself can be complex due to the presence of huge number of parameters affecting the decision. The super peer also introduces the problem of single point of failure because if it stops responding the entire cluster will go down.

The companies providing enterprise solutions for video conferencing like Vonage [1] rely on conference controllers like MCU or SFU which are deployed on the server. Using a SFU to forward the streams increases the reliability and reduces the load on the peers in

[1] https://www.vonage.com/

the network. The server can selectively encode the incoming stream into multiple streams and send them based on resolution and bitrate capabilities of the peers (Petrangeli et al.; 2018). So, even less powerful devices like mobiles and tablets can be connected to the network as all the heavy lifting is being done by the server. Petrangeli et al. (2018) proposes a dynamic framework that automatically re-encodes the stream based on the the bandwidth capabilities of the connected users. This ensures every user receives the best possible high quality audio/video streams improving the Quality of Expereince (QoE) of the application. However, since it is a centralized model handling multiple users in multiple sessions, scaling such a system can be very expensive.

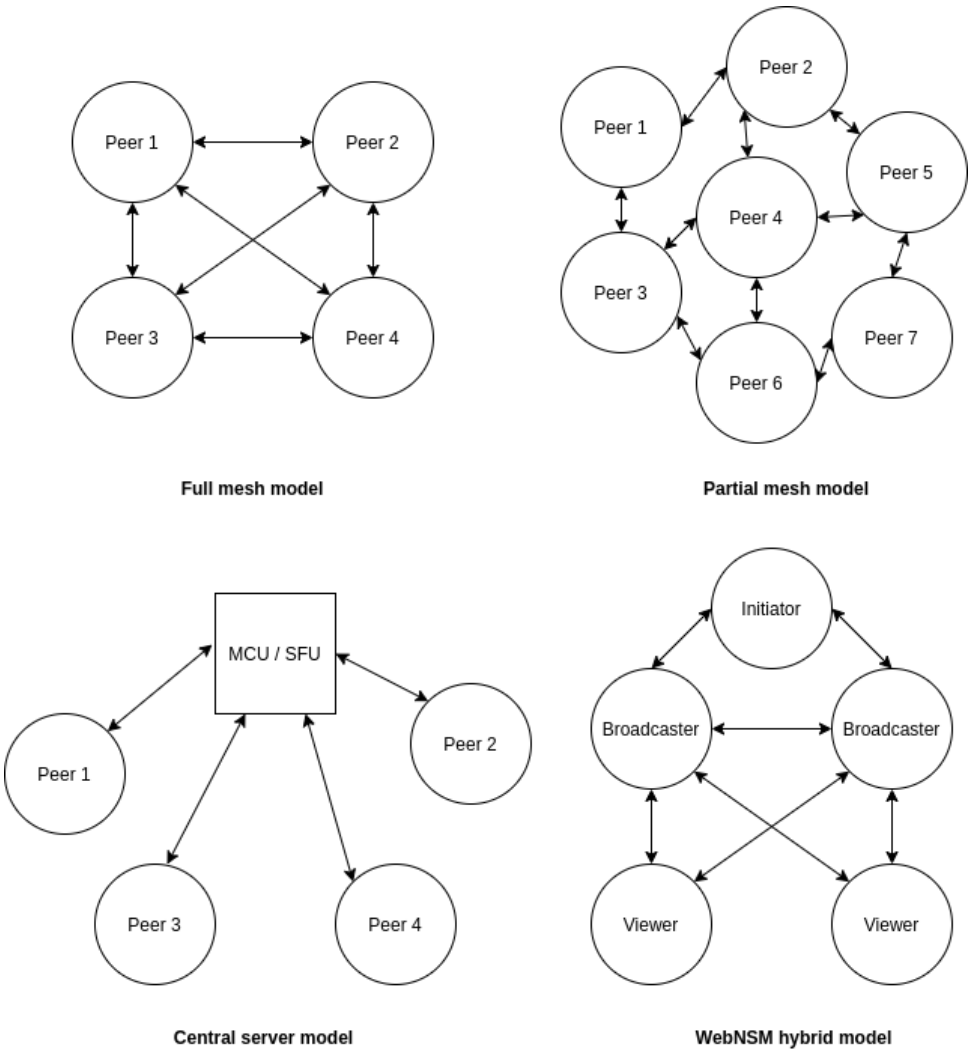


Figure 1: Different WebRTC signalling mechanisms

Edan et al. (2018) have proposed a hybrid signalling model in their paper using the combination of star and mesh models named WebNSM. It works on the principle of assigning different roles to the user joining the network. There are three roles defined in WebNSM model which are initiator, broadcaster and viewer. The first peer connecting to the network becomes the initiator and it's responsibility is to forward the stream to any broadcasters in the network. The broadcaster then relays the streams to all viewers in the session. So, users with higher capabilities (CPU, bandwidth) can become broadcasters to improve the performance of the network. However compared to the strategy used by

El Hamzaoui et al. (2018) which dynamically calculates the capabilities of the user and selects super peers, WebNSM does not take into account any changes in the network. So, if there are any failures in the nodes specifically if any broadcaster node goes down, the performance of the model will be impacted. The hybrid model however far outperforms the classical mesh network as shown in their results (Edan et al.; 2018). Our proposal relies on geolocation of the nodes which has lower chance to change to have any impact on the network. It also provides better reliability when compared to WebNSM as every node is connected to N(reliability factor) other nodes reducing the chances of failures in the network.

The different models that were discussed are presented in the above Figure 1.

A comparison of all the above signalling mechanisms based on different factors of implementation complexity, reliability and cost is shown in the below table Edan et al. (2019):

| Network | Scalable | Complexity | Reliability | Cost | Research |
|---|---|---|---|---|---|
| Mesh/Star | No | Simple | Very High | Low | Edan et al. (2018) |
| Tree/Graph | Yes | Medium | Low | Medium | Elleuch (2013) |
| Supernode | Yes | Complex | High | Medium | Paik (2015) |
| Centralized MCU/SFU | Yes to an extent | Complex | High | High | Petrangeli et al. (2019) |
| This paper | Yes | Medium | High | Medium | - |

## 2.2 Distributed architecture

The dynamic conference controller using a smart SFU proposed by Petrangeli et al. (2018) works very well for a medium sized network. However, if it has to scale to huge number of sessions each with many peers connecting to it then the single SFU model will find it difficult to manage so many sessions. With the availability of cloud based scalable architecture, there have been many proposals to overcome this issue. The SFU spends lot of its compute resources on dynamic re-encoding of the streams for different users which can be processed in parallel by distributing the load. Zakerinasab and Wang (2015) proposes a distributed solution which can split the video stream into multiple chunks and process them in parallel for better performance. The framework proposed by Zakerinasab and Wang (2015) makes use of a dependency-aware algorithm by analysing the visual features of the video which splits the stream into variable chunks. The visually similar frames are thus placed in one block and processed faster. With such an approach, the media streams can be processed without having a load on the controller thus increasing the scalability of the entire system. However, the integration between the conference controller and a distributed trans-coding algorithm introduces high implementation complexity and a lot of moving parts which reduces the reliability of the system. Another alternative to this solution can be to make a scalable SFU which can be interconnected with other SFUs in a cluster to forward the streams.

Rodríguez et al. (2016) have developed a scalable micro MCU architecture which is designed to relay streams to the users in a network which is managed by a control layer. The micro MCU is called as OneToMany(OTM) unit which act as a forwarder for audio and video streams. A single node can be connected to multiple OTMs and receive streams

from any number of OTMs but it can send it's stream to only one OTM. These atomic media units i.e OTM as specified in the paper can be created on-demand based on the requirement of the resources. As supported by their results, this distributed architecture indeed provides better performance and scalability. However, the management of the cluster and efficiency of the micro MCUs deployed in the cluster needs improvements which is also mentioned in their possible future lines of work.

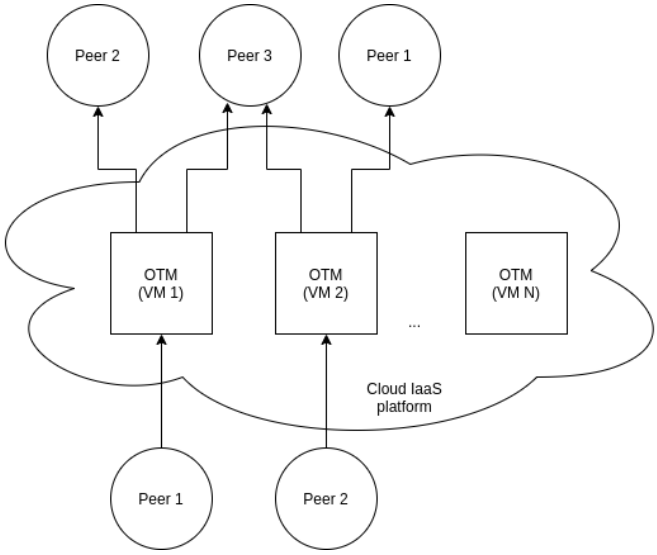


Figure 2: Distributed micro MCU using OTM Rodríguez et al. (2016)

The distributed architecture proposed by Rodríguez et al. (2016) is shown in the Figure 2. All the peers shown belong to a single session where Peer 1 and Peer 2 are sending the audio/video data to the OTM. The OTMs then relays these streams to all peers connected in the network. The solution proposed by Rodríguez et al. (2016) is a very good fit for offering a Platform as a Service (PaaS) since it is highly scalable and provides good performance. However, maintaining the entire cluster and launching multiple instances to manage the OTMs can be expensive. Feng et al. (2012) have designed a similar distributed solution over the cloud but unlike the proposal by Rodríguez et al. (2016), the data centers or servers do not directly communicate with the peers. They form a interconnected network in the cloud which forwards the stream internally to the server which is closest to the peer first. The closest server then forwards this stream to the peer thus reducing the network travel costs considerably. Their results show a considerable improvement in performance when compared to pure peer-to-peer solution. This design is similar to our approach where the peers are connected based on their geographical location but the complexity in our approach is much less as there is no interconnected server architecture required.

Wu et al. (2013) have proposed a scalable architecture in the cloud which is specifically developed to allow low powered mobile devices to join a video conference and send/receive streams at high quality. They use surrogate servers to achieve this by connecting each peer to a server in the cloud. All the processing is therefore done remotely in the cloud for the device thus enabling low powered devices to receive high quality streams. The Figure 3 shows the model of vSkyConf which is proposed by Wu et al. (2013) in their paper.

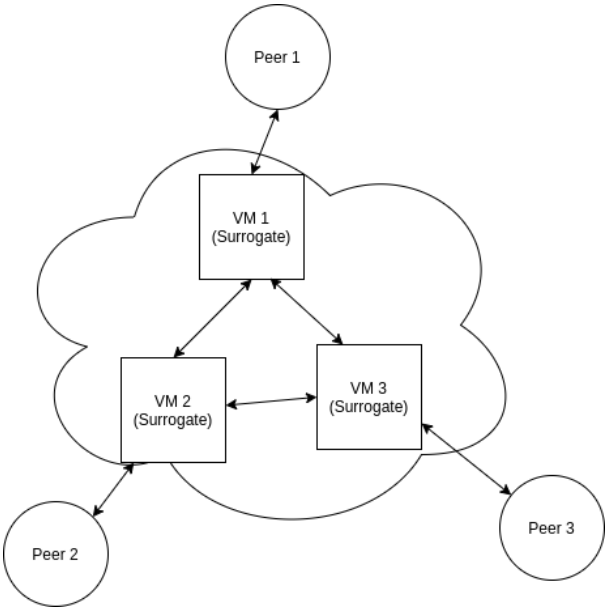


Figure 3: VSkyConf Wu et al. (2013)

The surrogate servers are launched in their own VMs in the cloud and are connected to each other in a mesh model. The surrogate server encodes the stream into optimum bitrates which can be handled by the peers. It also keeps a buffer store so that if there are any disruptions or lags in the network the stream transfer is not affected. The surrogate servers itself communicate with each other using optimal paths to decrease latency based on a heuristic approach which is part of their algorithm. Although it provides a scalable solution, assigning a surrogate server for every node in the network can lead to unnecessary network hops and higher bandwidth costs. Compared to distributed OTM architecture proposed by Rodríguez et al. (2016), spawning VMs for every user might not be feasible for larger video conferences.

The partial mesh model proposed in this paper has the potential to integrate with any of the above cloud based scalable solutions thus allowing it to scale globally. This also aligns with the future line of work described by Rodríguez et al. (2016) where they talk about deploying the OTMs based on proximity of the connected nodes in the network.

## 2.3 Bandwidth optimization

There has been extensive research to optimize the media transmission process to save bandwidth of both client and the server. Grozev et al. (2015) propose an intelligent SFU design which works on user activity in the session. It finds out N most active users and sends the streams to these nodes. This can work well for an audio conference where lack of activity can be replaced by not sending the media unnecessarily but it is not suitable for other use cases where the media needs to be sent to all the users all the time. Another approach to increasing the efficiency of media transmission is detection of movement of the the user who is sending the video. If the person is a broadcaster and there is not much motion detected, the spatial and temporal resolutions can be fine tuned to reduce the media data transferred between the peers and still maintain the quality of the stream. For example, if the video shows very fast movements, some frames can be skipped without affecting the quality of the video (Bakar et al.; 2019). Although this improves the efficiency of stream exchange, the process of dynamically trans-coding the stream is

expensive and not every device may be capable of doing it without affecting the Quality of Experience(QoE) of the application. It is possible to offload this process to SFU on a server but it will defeat the original purpose of reducing bandwidth consumption.

Although the previous section proposes many solutions for scalability by launching clustered servers there is a lot of unnecessary bandwidth transfer due to sub-optimal sever communication. Hu et al. (2016) propose a solution to this problem in their paper by describing an algorithm to optimize this communication by selecting servers based on distance from the peers. Their selection algorithm works on finding the distance between the physical location of the server and the geolocation of the users joining the session. This algorithm can be applied to some of the previous proposals (Wu et al.; 2013; Rodríguez et al.; 2016) to further improve their results. Their solution follows the same approach as given by Feng et al. (2012) where the server agents are connected to each other in a mesh topology. There is a latency discovery phase where every user trying to join to the network is tested based on the ping with each server. Using a heuristic based approach, the closest or the most optimal agent is selected for the peer. Although this approach selects the best server saving bandwidth and improving QoE of the application, ping latency is not very reliable and the results might change over time. A solution can be to run this algorithm every few intervals to get new optimal servers and if there is big difference connect the peer to this new server. The solution described by Hu et al. (2016) is similar to the approach taken in this paper but their architecture was designed for a peer to server agent network. On the contrary, our approach optimizes the peer-to-peer network using the location data of the nodes.

# 3 Methodology

After reviewing the literature, many different types of signalling mechanisms were found which tried to optimize video conferencing and broadcasting solutions. Edan et al. (2018) discussed about the different hybrid models and how they suited different use cases. After looking at the different available models, the partial mesh model was created which is based on the work by Feng et al. (2012). They created an interconnected network on the server, the proposed partial mesh model on the other hand creates a similar network on the client side along with improvements based on geolocation.

The research was developed in four different stages as shown in the Figure 4

## 3.1 Partial mesh model with N connections

We first needed a starting point for our project to apply the proposed optimizations. There are many sample WebRTC based open source projects available on GitHub. The video broadcasting library developed by Muaz Khan[2] was selected as the base project because it already had implemented a similar scalable mesh model.

The signalling mechanism used in this library was then modified to connect to N(reliability factor) other nodes. This ensured higher reliability since if one node went down, it would still have N-1 nodes from which it could display the stream to the user

[2]https://github.com/muaz-khan/RTCMultiConnection

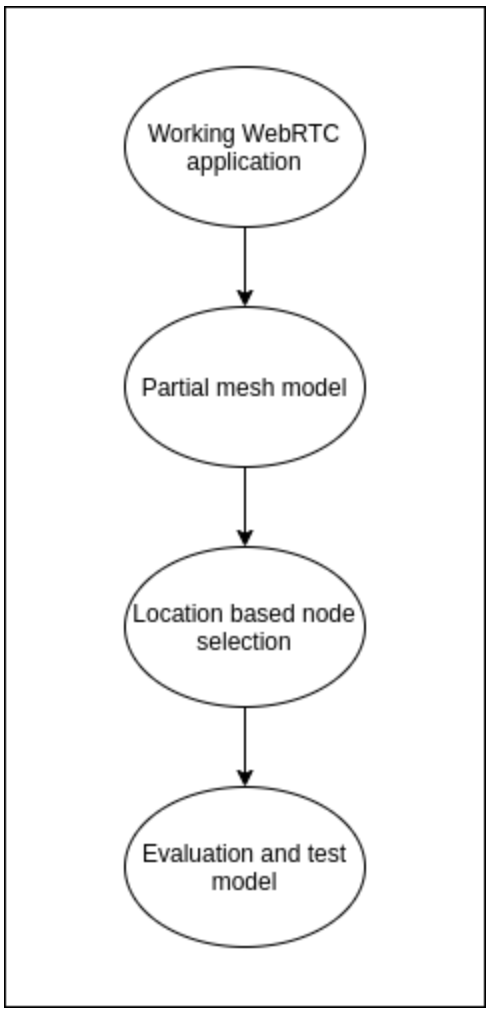


Figure 4: Project development

## 3.2 Selecting closest nodes based on location

The nodes connecting to each other did not have any logic in their selection. To select the closest node, HTML5 Geolocation API was used to retrieve the user's location which was sent to the signalling server during initial connection establishment. The selection algorithm was then modified to calculate distance between every node to connect to the closest N nodes in the network.

## 3.3 Testing using Pupeteer and Chrome

The optimizations were working fine locally but in order to test the performance under load, we needed to create users using code instead of manually launching browsers using UI. The first option was testRTC as it allowed you to launch multiple browsers and run custom scripts on them. However it could not be used as it was an enterprise only offering with the free version only providing with maximum concurrency of two browsers. There were many test frameworks that were tried like KITE, Jattack, WebdriverIO, WebdriverRTC, Pupeteer and Pupeteer cluster. Some frameworks were either not publicly available, not updated or were overly complex for our use case. Pupeteer proved to be the best framework for launching headless browsers and it also allowed to pass any kind

of configurable developer options during launch. The launch options allowed use of fake media streams and location simulation was possible using Pupeteer page evaluate helper methods

### 3.4 Fetching statistics and analysis

Chrome provides an internal UI when you enter the URL **chrome://web-internals** in the address bar. This interface allows you to analyse all connected streams in a WebRTC call. It provides many graphs and outputs to calculate different performance metrics. These metrics are also available with the getStats() call to the peer connection object. These metrics were used to plot graphs using HighCharts library.

## 4 Design Specification

### 4.1 Project working

The sequence of how the peer network is established is explained in the Figure 5

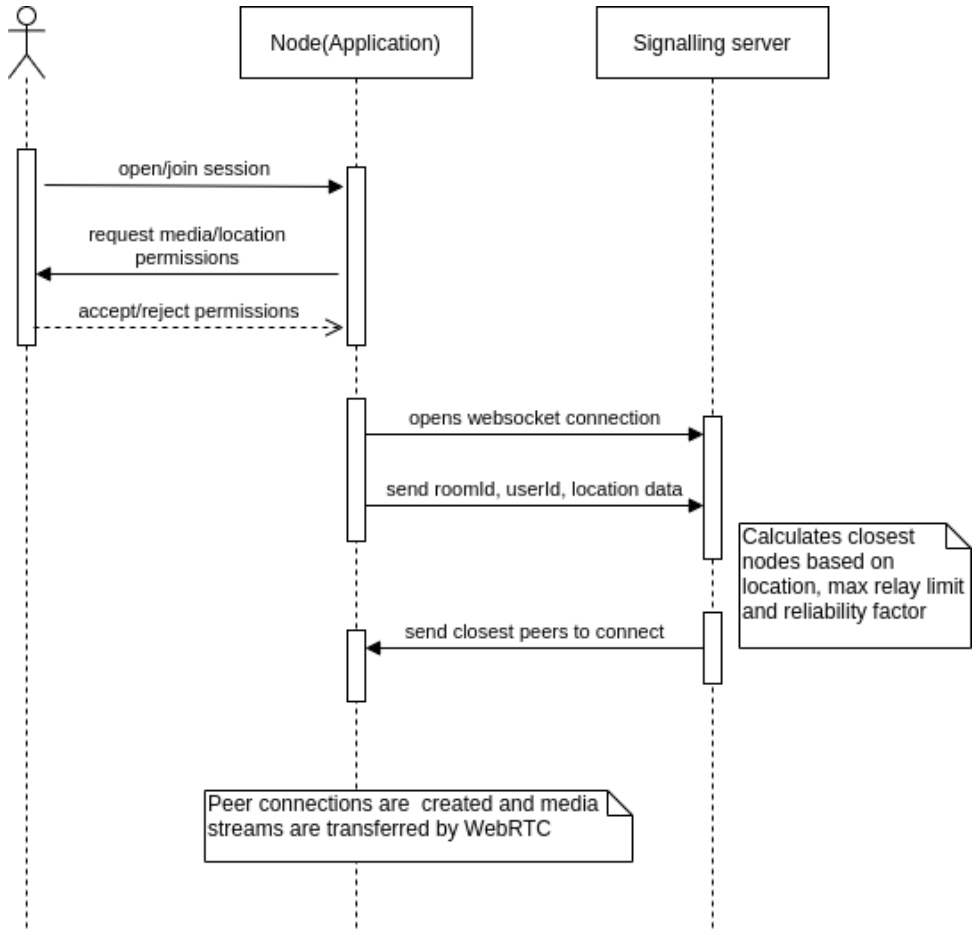


Figure 5: Flow diagram

The user first loads the application by visiting the web page and requests to create a new session or join an existing session. The application which acts as a node requests permission from the user to capture the user media and the current location data. If the

user accepts these requests, the node then goes ahead and establishes a socket connection with the signalling server to send this data. The node then passes on all details required to either create a new session or join an existing session. The signalling server has information of all the nodes in the session which it uses to calculate the most optimum nodes to connect to and returns that data back to the application. The node or application then initiates peer connections with these nodes using the WebRTC APIs.

## 4.2 Working of partial mesh model

The main idea behind partial mesh model is to avoid unnecessary connections that are made in a full mesh network. Since the same stream is duplicated in a mesh network, we can instead relay it once to the users by forwarding the stream through every node. However, this introduces multiple points of failures since when a single node goes down, all the children nodes that it was relaying the media to, will stop receiving the stream. To solve this issue, two parameters are introduced:

- **Reliability factor (for receiving streams)**
  Every node in the network is connected to at least N(configurable) other nodes. The network acts like a full mesh model until there are N nodes in the network. A higher value of N can thus increase the reliability of the network.

- **Max relay limit (for forwarding streams)**
  This is the maximum limit of relays that a node can send the stream to. A higher number allows a node to send the stream to more nodes which might decrease the reliability of the network but that also means there is a possibility of closer nodes to connect to it which can increase the performance of the network.

## 4.3 Signalling mechanism

The selection algorithm for selecting the most optimum nodes in the network is given below:

**Algorithm 1** How peers are selected from a network

```
Data: N = Max number of nodes to which node can forward its stream
R = Number of connections each peer should have for redundancy
G(P) = Function to find number of connections a peer has made
while peer P exists in list with G(P) < N do
    if number of relays for P >= R then
        skip this peer
    else
        find distance between P and current peer and add to list
    end
end
sort the nodes based on distance closest to peer
return first R nodes from this list
```

HTML5 Geolocation API is used to identify the closest nodes in the network. When the peer first connects to the signalling server using sockets, it sends its latitude and longitude values to the server. The server stores this location data for every user. When

the node sends a connection request to the signalling server, it will use the **Haversine** formula to calculate the great-circle distance between the two points of (latitude, longitude). It will then select N closest nodes to the requesting user and send their ids back. The peer is then able to connect to these closest nodes using the WebRTC API.

# 5 Implementation

## 5.1 Software frameworks and technologies

There are three main projects that were developed/modified for this research:

- **Implementation of partial mesh model**
  This was the main project written using Node.js and socket.io for the backend and HTML/javascript for the frontend. It includes the partial mesh model implementation as part of a NPM library. The reliability and location calculations for selecting closest nodes are written as part of this library. The frontend also includes script for generating statistics using WebRTC APIs and graph generation using HighCharts library.

- **WebRTC broadcasting using central server model**
  This was an existing project cloned from Github [3] for doing a comparison based test against the proposed solution. It was also modified for statistics generation using WebRTC APIs and HighCharts library.

- **Pupeteer based test project**
  This project was written from scratch using Pupeteer to emulate nodes in the network by launching headless browsers in the server. It makes use of several developer options provided by chrome to fake media and simulate geolocation. Many scripts are written to throttle network based on location and launch any number of nodes in parallel from the command line.

## 5.2 Signalling implementation

The signalling communication first happens over web sockets between the signalling server and the nodes. When the application is loaded, the user is asked for permission to capture the media using WebRTC media APIs and permission to retrieve the current location using the HTML5 Geolocation API. The data is then sent to the signalling server which already has data of all existing peers in the network. It returns the ids of the nodes that it needs to connect to based on max relay limit, reliability factor and geolocation distance of the nodes in the network.

## 5.3 Output statistics and graphs

A separate project was written for creating the test environment. Pupeteer script was implemented that launched headless chrome browser with certain developer options to

[3] https://github.com/Basscord/webrtc-video-broadcast

join the video broadcast without any manual intervention. This was possible by –**use-fake-ui-for-media-stream** which would make it possible to capture fake media stream without asking for user permissions. These headless browser sessions were launched on multiple EC2 instances. A **t2-medium** instance can support up to 20 parallel browser sessions. To generate the graphs for the outputs, the statistics provided by WebRTC API **getStats()** is used. It provides an asynchronous way to fetch metrics of all peer connections for the node. These metrics were aggregated and displayed on line graphs using the HighCharts plotting library.

# 6 Evaluation

The focus of this paper was to optimize the peer-to-peer model for video broadcasting so it was necessary to compare the partial mesh solution with some other solution to analyse the performance. The central server model where the central peer acts as a broadcaster was selected as this model suits very well for one-to-many stream broadcasting. The test was performed by first connecting the broadcaster to the session then slowly adding viewer peers every 10-15 seconds from an EC2 instance. The bandwidth statistics were compared for both broadcaster and viewer.

## 6.1 Central server model

In a central server model, the broadcaster peer acts as the single distributor node in the entire network. It sends the stream to all the peers connected in the network. As seen below in the Figure 6, the *BytesSentPerSecond* shows an increasing trend with every new user added to the session.

### 6.1.1 Broadcaster perspective

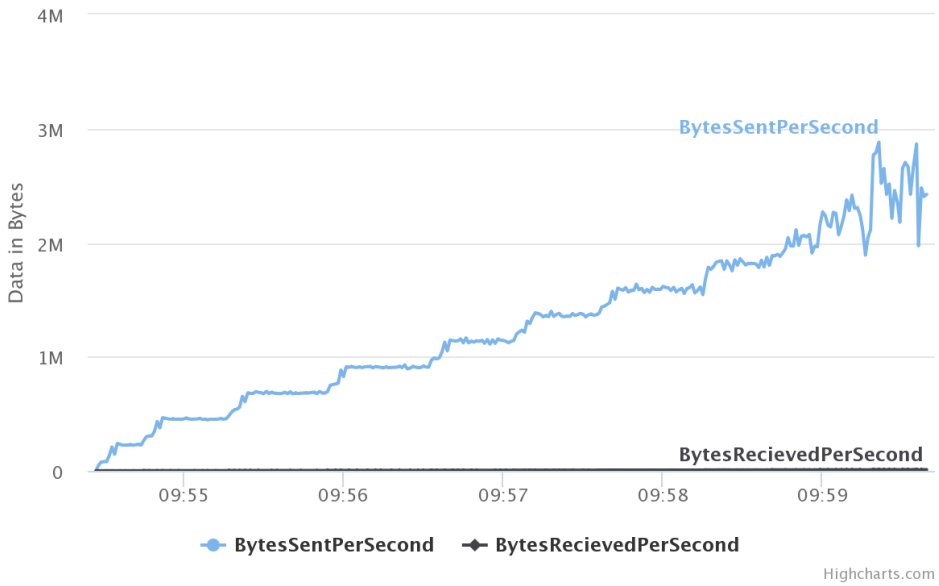


Figure 6: Broadcaster bandwidth statistics

The graph(Figure 6) shows a linear increase in bytes sent per second every time a new user is added and no changes in bytes received per second over the entire session. When the number of receivers reach 15 (in this case) at around 10:00 hours in the graph there is a lot of fluctuation in the bandwidth. This is where the single peer is unable to keep up with the bandwidth requirements from all peers in the network. Any new viewers in the session may lead to packet loss or lag in the video streams.

### 6.1.2 Viewer perspective

The viewer in this model only receives the stream from the broadcaster so we should see a constant bandwidth consumption throughout the session which is reflected in the graph(Figure 7) below. There is no change seen in the bandwidth until the number of users reach a high enough number where we again see the same fluctuation that was observed in the broadcaster perspective. The broadcaster is no longer able to send streams at a constant rate so the *BytesRecievedPerSecond* starts showing some dips at the end. The viewer is not sending stream to any other users so the *BytesSentPerSecond* is always 0.

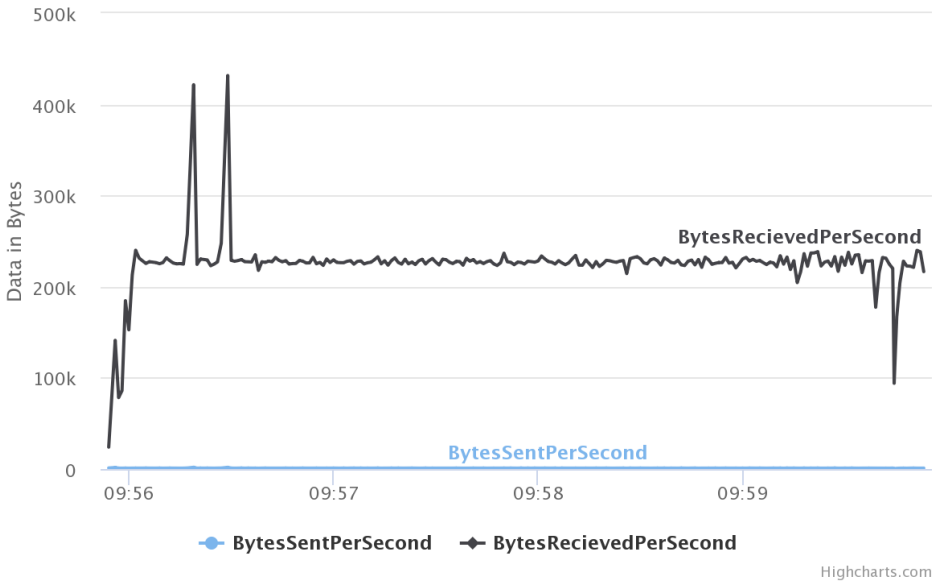


Figure 7: Viewer bandwidth statistics

## 6.2 Partial mesh model

### 6.2.1 Broadcaster perspective

Similar to the partial mesh model, here the broadcaster first joins the session and then slowly new viewers are added to the session every few seconds until the number of viewers reach 15. The Figure 8 shows the perspective of the broadcaster in the partial mesh model. When compared to the Figure 6, even at the maximum number of viewers, the *BytesSentPerSecond* remains constant. The central server model was sending millions of bytes i.e around 2 to 3 Megabytes (MB) of data every second whereas here the maximum data sent per second is around 500 Kilobytes (KB). As seen in the figure, initially the *BytesSentPerSecond* increases then it becomes constant when it reaches a certain limit (no. of users) set in the code after which the stream is relayed using the other viewers

in the network. Since this node is the broadcaster, bytes are not received from any other peer in the network.

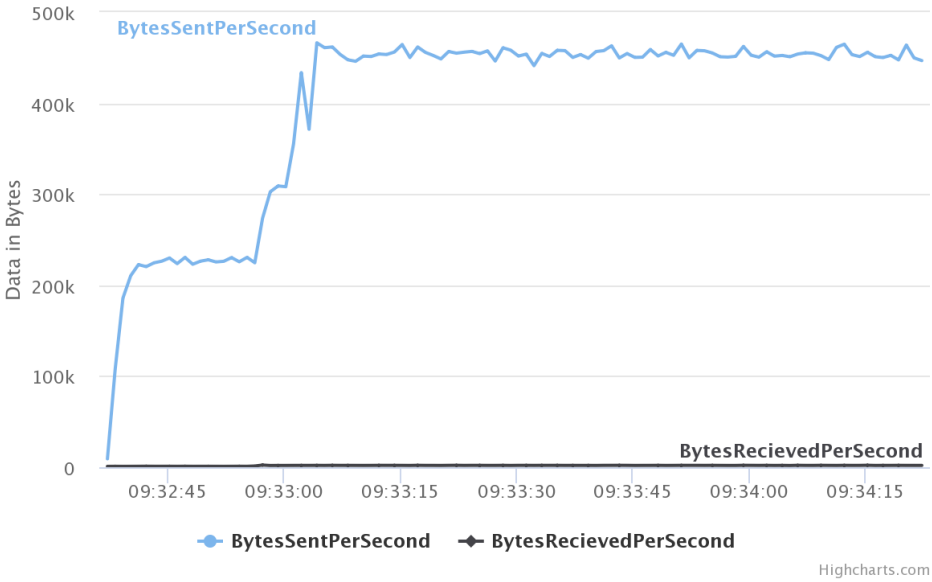


Figure 8: Broadcaster bandwidth statistics

### 6.2.2 Viewer perspective

The viewer receives the stream from broadcaster and also acts as a relayer to forward the stream to other peers when the number of users in the session increases above a set limit. The Figure 9 shows that the viewer, similar to the central server model has a constant rate of *BytesRecievedPerSecond*, but unlike it after few users are added in the session, it also starts sending the stream to other users. This can be seen from the upward trend in the graph for *BytesSentPerSecond*. Again, similar to the broadcaster, it rises and then reaches a constant value at around 500 KB per second.

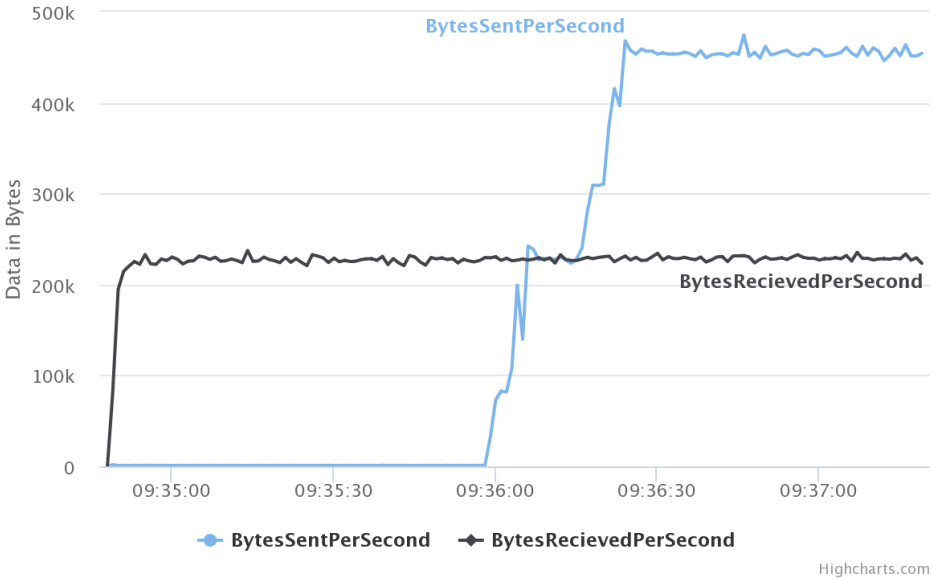


Figure 9: Viewer bandwidth statistics

## 6.3 Verify reliability and location using simulation

### 6.3.1 Reliability

We follow the same testing approach as done previously but this time we set the reliability factor as 2 so that the viewer nodes receive streams from multiple nodes. As shown in Figure 10, we can see that the node is receiving 400 KB of data compared to the previous tests where the nodes only received half of that. The viewer node is thus receiving the streams from multiple peers based on the reliability factor set in the code.

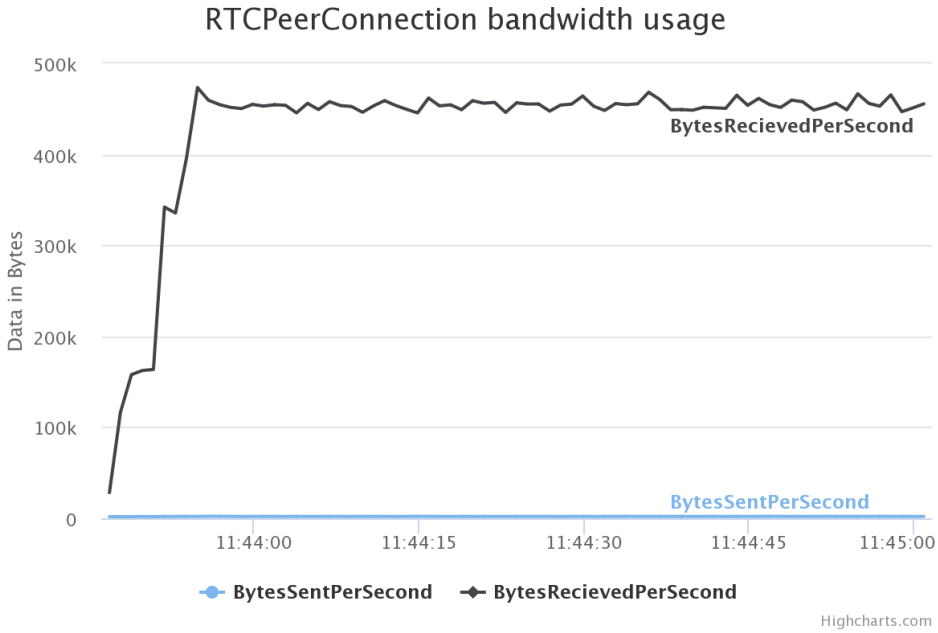


Figure 10: Viewer bandwidth statistics

### 6.3.2 Geolocation simulation

For verifying whether our model is connecting to the closest available nodes we launch browsers with simulated locations across the globe and then locally launch a viewer and see the data about the peers connected to this viewer. As seen in the Figure 11, we can see latitude and longitude location data of two nodes printed in the console logs. The viewer is connected to nodes in Dublin and UK which are closest to the local running node which was in Dublin. The other nodes that were launched during the test were spread across Europe and USA.

```
> connection.peers.forEach(peer => {
      console.log(peer.extra.location)
  });
  ▶ {latitude: 53.35014, longitude: -6.26615}
  ▶ {latitude: 51.509865, longitude: -0.118092}
```

Figure 11: Location logs

## 6.4 Discussion

The above results show that the partial mesh solution saves significant bandwidth thus performing better than a central server model for video broadcasting. When the session has 15 users, the broadcaster saves almost 80% of bandwidth in the partial mesh model. There are no benefits for the viewer node in terms of bandwidth since in both models there is no huge media exchange even for large sessions.

However in our proposed solution as the viewer also acts as a relayer and is therefore sharing the stream the reliability of the model decreases as there are now multiple points of failures. To overcome this, the reliability factor can be increased which ensures that nodes creates multiple connections and receives streams from more than one peer. This factor increases reliability but at the same time has an impact on the bandwidth. As seen in Figure 10, the bandwidth usage increases by 200KB/s when the reliability was increased to 2. This however is still way less than the bandwidth consumption of a pure mesh model as the peers would be connected to every single node in the network (Edan et al.; 2018).

The nodes in the network are connected based on the location proximity to other nodes which should improve the transfer rates of streams over the network. The Figure 11 shows that the peers are connected to the closest nodes based on geographic distance between each other which should improve the performance of the network (Rodríguez et al.; 2016). The location based simulation was run but it was not possible to get accurate performance metrics to compare it with any existing solution. If some QoE metrics could be added to the test statistics, it would be possible to find out exactly what contribution does the location optimization is making to improving the QoE of the session. However, the main focus of this paper was improving the scalability of the mesh model without compromising on its reliability which is achieved as per the above results.

# 7 Conclusion and Future Work

The partial mesh model proposed in this paper was successfully implemented, tested and verified. The research question stated was to see if it was possible to create a scalable peer-to-peer model for video broadcasting. The solution and the results show that the model is indeed scalable however it cannot be scaled to unlimited nodes as the number of network hops between the nodes will increase with the number of peers. The research results show 80% bandwidth reduction for broadcaster node in the network when compared to standard central peer broadcasting model. The reliability factor used shows that a peer is able to receive multiple streams from other nodes in the network which increases the reliability of the network. The model also ensures that closest peers in the network are connected to each other which should increase the quality of experience of the application. There is no improvement in the viewer nodes in terms of bandwidth reduction but the proposed model is much more scalable when compared to the standard broadcasting model.

The research was focused on locally optimizing the peers in a small geographical region because peer-to-peer connections across large distances may lead to unreliable communication. To fix this issue and to provide even higher scalability, this solution can be integrated with a cloud based architecture where the server itself acts as a peer and connects to the network. Any long distance communication will be going through the servers using their reliable and fast network and the servers can be scaled horizontally to offer infinite scalability potential to the service. Another line of improvement which is possible is to make the max relay limit factor dynamic so that the nodes closer to the broadcaster have a higher relay limit than the nodes far from the broadcaster. This will reduce the number of network hops in a larger broadcasting session.

# References


Bakar, G., Kirmizioglu, R. A. and Tekalp, A. M. (2019). Motion-based rate adaptation in webrtc videoconferencing using scalable video coding, *IEEE Transactions on Multimedia* **21**(2): 429–441. JCR Impact Factor: 3.977(2018).

Blum, J., Booth, S., Gal, O., Krohn, M., Len, J., Lyons, K., Marcedone, A., Maxim, M., Mou, M. E., O'Connor, J. et al. (2020). E2e encryption for zoom meetings.

Carlucci, G., De Cicco, L., Holmer, S. and Mascolo, S. (2016). Analysis and design of the google congestion control for web real-time communication (webrtc), *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, Association for Computing Machinery, New York, NY, USA.
**URL:** *https://doi.org/10.1145/2910017.2910605*

Dutton, S. (2013). Webrtc in the real world: Stun, turn and signaling, *Google, Nov* .

Edan, N., Al-Sherbaz, A. and Turner, S. (2018). Design and implement a hybrid webrtc signalling mechanism for unidirectional & bi-directional video conferencing, *International Journal of Electrical and Computer Engineering* **8**: 390–399. JCR Impact Factor: 0.368(2018).

Edan, N. M., Al-Sherbaz, A. and Turner, S. (2019). Performance evaluation of resources management in webrtc for a scalable communication, *in* K. Arai, S. Kapoor and R. Bhatia (eds), *Intelligent Computing*, Springer International Publishing, Cham, pp. 648–665.

El Hamzaoui, A., Bensaid, H. and En-Nouaary, A. (2018). A new signaling topology for multiparty web real-time video conference networks, *Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications*, SITA'18, Association for Computing Machinery, New York, NY, USA. Core Ranking: B.
**URL:** *https://ezproxy.ncirl.ie:2079/10.1145/3289402.3289517*

Elleuch, W. (2013). Models for multimedia conference between browsers based on webrtc, *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Lyon, France, pp. 279–284. Core Ranking: B.

Feng, Y., Li, B. and Li, B. (2012). Airlift: Video conferencing as a cloud service using inter-datacenter networks, *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–11. Core Ranking: A.

Grozev, B., Marinov, L., Singh, V. and Ivov, E. (2015). Last n: Relevance-based selectivity for forwarding video in multimedia conferences, *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '15, Association for Computing Machinery, New York, NY, USA, p. 19–24. Core Ranking: A.
**URL:** *https://ezproxy.ncirl.ie:2079/10.1145/2736084.2736094*

Hu, Y., Niu, D. and Li, Z. (2016). A geometric approach to server selection for interactive video streaming, *IEEE Transactions on Multimedia* **18**(5): 840–851. JCR Impact Factor: 3.977(2018).

Jansen, B., Goodwin, T., Gupta, V., Kuipers, F. and Zussman, G. (2018). Performance evaluation of webrtc-based video conferencing, *SIGMETRICS Perform. Eval. Rev.* **45**(3): 56–68.
**URL:** *https://doi.org/10.1145/3199524.3199534*

Ng, K.-F., Ching, M.-Y., Liu, Y., Cai, T., Li, L. and Chou, W. (2014). A p2p-mcu approach to multi-party video conference with webrtc, *International Journal of Future Computer and Communication* **3**(5): 319.

Paik, J. (2015). Scalable signaling protocol for web real-time communication based on a distributed hash table, *Computer Communications* **70**. JCR Impact Factor: 2.766(2018).

Petrangeli, S., Pauwels, D., van der Hooft, J., Wauters, T., De Turck, F. and Slowack, J. (2018). Improving quality and scalability of webrtc video collaboration applications, *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, Association for Computing Machinery, New York, NY, USA, p. 533–536. Ranking: B3 (Source: Qualis).
**URL:** *https://ezproxy.ncirl.ie:2079/10.1145/3204949.3208109*

Petrangeli, S., Pauwels, D., van der Hooft, J., Ziak, M., Slowack, J., Wauters, T. and De Turck, F. (2019). A scalable webrtc-based framework for remote video collaboration applications., *Multimedia Tools and Applications* (6): 7419. JCR Impact Factor: 2.101(2018).
**URL:** *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=edsgaoAN=eds*
*livescope=sitecustid=ncirlib*

Rodríguez, P., Álvaro Alonso, Salvachúa, J. and Cerviño, J. (2016). Materialising a new architecture for a distributed mcu in the cloud, *Computer Standards & Interfaces* **44**: 234 – 242. JCR Impact Factor: 2.441(2018).
**URL:** *http://www.sciencedirect.com/science/article/pii/S0920548915001014*

Uberti, J. and Jennings, C. (2013). Javascript session establishment protocol.
**URL:** *https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03section-1.1*

Vass, J. (2018). How discord handles two and half million concurrent voice users using webrtc.
**URL:** *https://blog.discord.com/how-discord-handles-two-and-half-million-concurrent-voice-users-using-webrtc-ce01c3187429?gi=475c02529a5c*

Wu, Y., Wu, C., Li, B. and Lau, F. C. (2013). VSkyConf: Cloud-assisted multi-party mobile video conferencing, *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, MCC '13, Association for Computing Machinery, New York, NY, USA, p. 33–38.
**URL:** *https://doi.org/10.1145/2491266.2491273*

Zakerinasab, M. R. and Wang, M. (2015). Dependency-aware distributed video transcoding in the cloud, *2015 IEEE 40th Conference on Local Computer Networks (LCN)*, Clearwater Beach, FL, USA, pp. 245–252. Core Ranking: A.