Thesis no: MSEE-2016:16

Implementation and Performance Optimization of WebRTC Based Remote Collaboration System

Improving Remote Collaboration

Venkesh Kandari

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering with Emphasis on Telecommunication Systems. The thesis is equivalent to 20 weeks of full time studies. The master thesis research was carried out at Ericsson AB and Semcon AB in Gothenburg, Sweden as part of the MERCO research project.

Contact Information:
Author(s):
Venkesh Kandari
E-mail: vekb15@student.bth.se

External advisor:
Ulrica Cullen
Innovation at Systems and Technology
Ericsson AB
Gothenburg, Sweden

University advisor:
Dr. Siamak Khatibi
Department of Communication Systems

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Web based multimedia applications has gained much importance over the past years because of its potentials and capabilities. In present market there are many applications available and are used by many companies. Over these applications WebRTC has grabbed much attention of the viewers. Being a web based application WebRTC can solve many issues like complex decision making, without any physical presence in corporate world. As a result, in future WebRTC play a key role in the remote meetings. However, while building a Web based real-time (WebRTC) application many challenges like scalability, cost-effective and interoperable needs to be considerd. The requirements and functions considered in thesis are taken form the user tests made in SEMCON and ERICSSON for research project Mediated Effective Remote Collaboration (MERCO).

The main objective of this thesis is to find architecture of WebRTC which suits the requirements of MERCO project. Selecting a signaling server which is having a low call start up time and has no intricacy in maintenance. Investigating the performance and scalability issues of the WebRTC architecture and to find an appropriate solution. Finally, to select a database that has to be used in the project, which should be more efficient and cost-efficient.

Before selecting the architecture, a fine research is done and Mesh architecture of WebRTC is used in MERCO project as it thoroughly meets the requirements like scalability, simplicity of implementing and no external server required for media streams. Face detection technique is proposed for constraining media streams. For signaling server XSockets and Node.Js servers are evaluated with respective of the call startup time between two clients using modern browsers chrome and Firefox. Databases are selected based upon the input and output response time taken by the OrigoDB and Azure Storage Table.

The final results show that XSockets when compared with Node.js is having low call startup time. Using face detection technique when media streams are constrained, it is observed that bandwidth and resource utilization is optimized. The input and output response time of the OrigoDB shows less when the data is below 500Kb.

Keywords: WebRTC, XSockets, Signaling, Scalability, Architecture Design, Performance Evaluation.

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my thesis advisor Dr.Siamak Khatibhi for his constant support throughout this research project. This thesis wouldn't be so steady without his valuable feedback and support.

I would also like to thank MERCO project managers David Gillblom (Semcon) and Ulrica Cullen for giving me this opportunity in MERCO project and also to the experts who participated for the validation of this research, without their participation this research could not be completed successfully.

I would also like to acknowledge Prof.Dr.Krut Tutschku of Blekinge Tekniska Högskola as the second reader of this thesis, and I am very gratefully indebted to his valuable comments on this thesis.

Finally, I would like to express my profound gratitude to my parents for giving work to me for unfailing backing me throughout my years of my study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

API	Application Programming Interface
DNS	Domain Name System
DOM	Document Object Model
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
FEC	Forward Error Correction
FPS	Frames per Second
HTML	Hyper Text Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IAT	Inter-Arrival Time
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IM	Instant Messaging
IoT	Internet of Things
IRC	Internet Relay Chat
ITU	International Telecommunication Union
JSEP	JavaScript Session Establishment Protocol
JSFL	JavaScript Flash Language
JSON	JavaScript Object Notation
LAN	Local Area Networks
MCU	Multipoint Control Unit
MEGACO	Media Gateway Control Protocol
NAT	Network Address Translation
OWD	One-Way Delay
QoE	Quality of Experience

QoS	Quality of Service
REMB	Receiver Estimated Maximum Bitrate
RFC	Request for Comments
RIA	Rich Internet Application
RTC	Real Time Communication
RTCP	Real Time Control Protocol
RTMFP	Real Time Media Flow Protocol
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
SCTP	System Control Transmission Protocol
SDP	Session Description Protocol
SFU	Selective Forwarding Unity
STUN	Simple Transversal Utilities for NAT
TURN	Traversal Using Relays around NAT
UDP	User Datagram Protocol
VoIP	Voice over IP
VVoIP	Video and Voice over IP
W3C	World Wide Web Consortium
WebRTC	Real Time Communications for the Web
WHATWG	Web Hyper Text Application Technology Working Group
WWW	World Wide Web
XSS	Cross-site scripting

1 INTRODUCTION

Videoconferencing, digital meetings and skype interviews have become prominent in today's workforce yielding the benefits of reducing time and money wasting as well as environmental impact due to travelling. Nevertheless, present technology still struggles to overcome challenges like scalability, integration, and interoperability to effectively support multi-party remote collaboration with tolerable user experience. Conventional video conferencing systems lack the capability of adapting to usage of system resources (CPU, memory, disk and network) along the fly[1]. Moreover, current technology is not able to capture, transfer and display important details like quick real-time feedback, hence complex collaborations like interactive team work still typically require physical presence of all partners involved. In order to optimize the development and business operations of enterprises, efficient and flexible solutions that support multi-part interactive RTC are required. The introduction of WebRTC has been highlighted as the solution to most of the challenges in RTC due to the flexibility of this technology, allowing lightweight RTC with cool features like screen sharing, screen recording, whiteboard sharing and video/audio chat over the web [2].

Web based conferencing such as webinars and webcasts supported over TCP/IP have been available in World Wide Web (WWW) since over 20 years ago. Since then, technology evolved from unidirectional to half duplex and full duplex media exchange, including the efforts to improve performance and reduce latency, data loss, utilization of resources like bandwidth and CPU through the use of advanced video codecs [3]. Affordable digital and video cameras on present day computers and smartphones are supporting the continuous increase in adoption of real-time internet video communication. All these positive efforts in real-time web communication together with increased demand for new applications with better quality experience and easy to integrate with WWW are some of the motivations behind WebRTC.

WebRTC is as an open source project with an effort to bring RTC APIs to JavaScript developers, enabling web applications to support different RTC functionalities such as voice calling, video chat, and P2P file sharing via web browser [4]. WebRTC APIs works through the conjunction of two technologies, Hyper Text Markup Language (HTML) and JavaScript [5]. Since it was released, effort is being put to integrate it with other communication technologies, frameworks and platforms for enabling rich, high-quality RTC applications to be developed for the browser,

mobile platforms, and IoT devices, to allow them all to communicate via a common set of protocols. WebRTC in combination of other technologies like XSockets, Node.js, Vconect, OpenCV, Kinect, Kurento and other APIs bring up a lot of attractive features and functions including real-time detection and recognition of faces, emotions, and speech and hand gestures straight from the browser [6] [7] [8]. Investigating the feasibility of right integrating these technologies with WebRTC can enhance the way RTC services are provisioned and implemented for variety of purposes. Remote collaboration within companies is a one area where WebRTC implementation is becoming common.

1.1 Overview

Dramatic outburst of internet communication is pushing for more human-centric innovations in telecommunication technologies to make life less complicated and more enjoyable for the people. Concurrent developments in the vision of Internet of Things (IoT) is forcing rapid deployment of advanced and efficient telecommunication solutions (5G, LTE, Wi-Fi, BLE etc.) for ubiquitous internet [9]. Digital communication over the internet is encouraging people to interact, talk, and conduct business meetings and share any kind of content in a more time efficient, more cost effective and more environmental and reliable and way [10]. Innovations, enterprises, business operations are now rapidly moving forward, credit to the introduction of low cost, reliable and simple platforms for real–time communication now available in browsers, WebRTC.

With connectivity, automatic intelligence and integrated cameras becoming standard features of our daily use devices, support for interactive real-time digital communication is becoming more and more realizable on laptops, smartphones, and tablets using different communication technologies. Now a lot of amazing real-time digital interaction features are now possible directly on internet browsers without any need of plugin download, which is Web Real Time Communication (WebRTC). WebRTC is bringing new features, functions and services that have never been available before [11] [12]. It is facilitating the possibility for people to have high-engagement and more intimate real-time digital communication over the internet. The possibility of experiencing friendly and informal communication is changing traditions and conventional communication habits, and transforming interpersonal relationships. Distance apart is no longer a problem to bring individuals and teams around the world

to connect, discuss and solve problems on a common task together. Now people efficiently connect remotely with colleagues, recruiters and friends in different ways like meetings, interviews, gaming or social networks. In addition, WebRTC solutions helps enterprises, businesses and companies to have lightweight communication alternatives that will increase the efficiency of their operations. WebRTC is also encouraging frontend designers, software and device developers to build interoperable, platform independent and multi-functional communication tools.

Industry partners are continuously looking to leverage the amazing RTC features that comes with WebRTC and are in the process of researching on and developing WebRTC systems for effective remote collaboration. Ericsson and Semcon are partners in an international research project, Mediated Effective Remote Collaboration (MERCO) which is aimed to improve remote collaboration in companies. The expected outcome of the project is a software product that enables and encourage interactive mobile teamwork and business meetings. The project is based on the idea that when a company is developing complex products, collaborative teams are required to achieve product development. Typically, different areas of expertise are required which could often be located in different cities and sometimes even countries. Systems and interactive aids that enable mobile communication in these user contexts are however lacking to provide an opportunity for participants to efficiently exchange ideas and feedbacks in all the stages of product development.

1.2 Problem Statement

Ericsson and Semcon are working on an international research project (MERCO) aimed to develop solution for improving remote collaboration within companies. By aiming for interoperable and platform independent solution, it highlighted that WebRTC in combination with suitable computer vision technologies will cheaply bring flexible interactive RTC. Building interactive multi-party RTC systems over the internet has always been characterised by consideration of the number of participants that can seamlessly join the conversation without significantly degrading the performance of the system. This is closely associated with the topology distribution of nodes that are being connected and more importantly the performance capabilities of different topologies tends to restrict the amount of nodes available for session.

With the flexibility of WebRTC technology, new RTC topologies are emerging. From the simple point-to-point topology to full mesh composition, choosing the right option for each application is very important to deliver a great user experience. Moreover, selecting the appropriate technologies to integrate with WebRTC for more interactive functionalities is also crucial for better performance of the system. This thesis focuses on the architectural and technical solutions to improve scalability of MERCO WebRTC based multi-party remote collaboration system with respect to specifications from meeting observation at Ericsson and Semcon.

1.3 Research Questions

1. What are the performance effects on WebRTC signaling technologies with respect to startup time in multiparty collaborations?
2. How face detection can be used to improve the performance of WebRTC conferencing systems based on resource utilization?
3. What is the effect on data base query execution time as the volume of data is increased in real time communications?

1.4 Hypothesis

1. For different WebRTC applications various technologies and architectures have impact on scalability
2. Using face detection technique, media streams are constrained. This results in the improvement of performance and scalability of the system.
3. Proper Database selection for a real-time application makes the system more efficient and saves resources.

1.5 Contributions

The outcome of this thesis will helps to find the appropriate architecture and signaling server to use for WebRTC different WebRTC applications which helps to implement in different real-time systems. It also guides to to find the various techniques to improve the performance, scalability and Network usage of WebRTC.

As the prototype is designed for corporate purposes, but source code can be used for different fields such as medical, automotive and marketing.

1.6 Split of Work

Some sections (section-1, 1.1, 1.2, 7.1) in this thesis document were written by fellow thesis worker Tinashe Kurehwaseka.

This thesis work is done in close collaboration with another thesis done by Tinashe Kurehwaseka at Blekinge Institute of Technology (BTH), along with other three thesis's done by Pengzhi Zhou at Chalmers Institute of Technology, Malin Nyström and Hedda Ottersten at Chalmers Institute of Technology, and Artem Gumeniuk at Gothenburg University.

Tinashe contributions helps in the completions of this thesis by providing a consistent technology, architecture for WebRTC and using of Emotion and speech detection techniques. Pengzhi's contribution helped to accomplish the implementations of the proposed solutions in this thesis. Malin, Hedda and Artem played a key role in designing the prototype and defining of the functional and non-functional requirements by conducting various user tests proposed in this thesis.

1.7 Methodology

In first part of this thesis gives a detailed history of the RTC and WebRTC background, fundamentals different architectures and components are mentioned. This mainly helps the developers to choose a best architecture and implementation method, suits to their RTC application.

Second part of this thesis describes three main experimental investigations, one is to choose an appropriate architecture to be used for MERCO project. Two evaluating the performance of the signaling technology and to improve the system scalability. The final investigation is done to choose a relevant database for the system.

1.8 Challenges

Considering that WebRTC is still under development and there is constant evolution in the WebRTC. The proposed solutions might only work in the modern browsers that supports WenRTC .

Considering that WebRTC is still under development while writing this thesis, some of the statements may change in the later versions of WebRTC. Even though this doesn't affect much on conclusions.

1.9 Outline of thesis

This thesis is structured as follows:

Chapter 2: walks through the Background of the RTC and WebRTC and also explains various WebRTC architectures and WebRTC components. Briefs the different databases that can be used in RTC applications.

Chapter 3: Explains the related works that are carried out in the WebRTC and RTC

Chapter 4: Describes the research carried out in this thesis and also explains the methodology for experiments.

Chapter 5: Details the experimental setup and procedure carried in this thesis. It also details the overcome of the challenges and the techniques used in this thesis.

Chapter 6: Presents the results of the Experiments.

Chapter 7: Discusses the results of the experiments that are performed in this thesis.

Chapter 8: Conclusion

2 BACKGROUND WORK

2.1 Real-Time Communication

Real time communication is an evolving technology these days. Real-Time communication allows the users to exchange the data and media streams in real time. Now-a-days many companies are looking forward to have their meetings in real time, so that many critical challenges can be achieved through communication. It can save many resources, it is also effective when compared to the existing technologies.

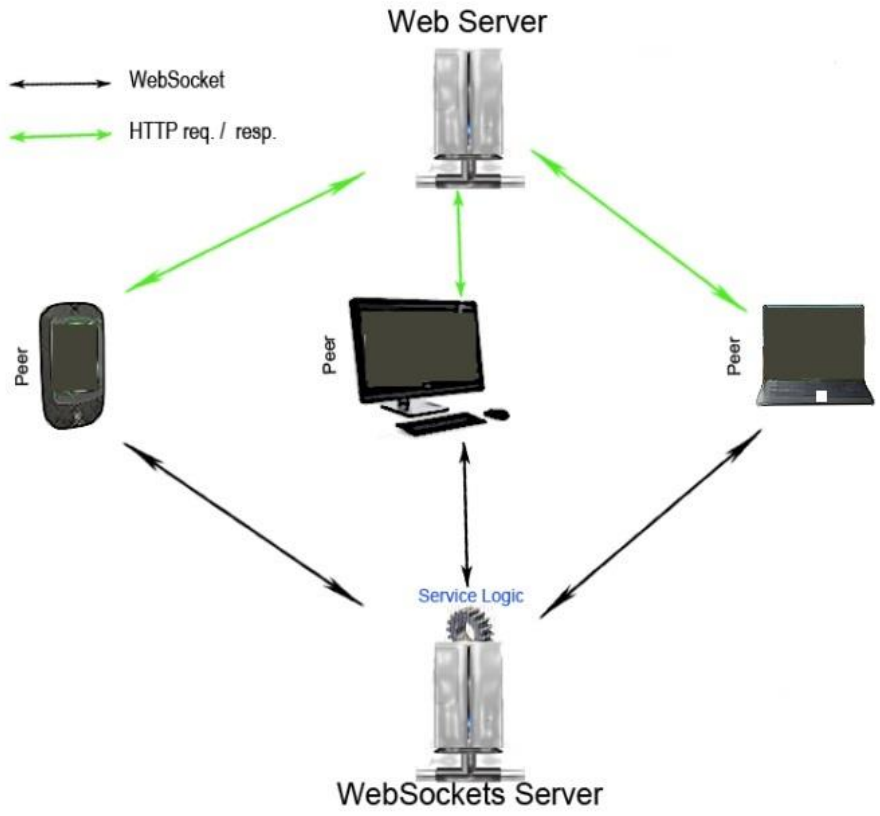

Figure 2.1: General Architecture of Real-Time Communication[13].

From the above figure it is shown that the peers send http request to the web server and these requests are identified in the websockets sever, then the response is sent back to the identified peers.

One of the most popular technology existing in today's Real time communications is WebRTC. WebRTC and its types of architecture are discussed in later sections.

2.2 Real-Time Communication in WebRTC

WebRTC is as an open source project with an effort to bring RTC APIs to JavaScript developers, enabling web applications to support different RTC functionalities such as voice calling, video chat, and P2P file sharing without requiring any plugins. APIs can be defined as a collection of methods and callbacks that help developers to access available technologies for specific functions. WebRTC APIs works through the conjunction of two technologies, Hyper Text Markup Language (HTML) and JavaScript. HTML is the format for serving web applications, and JavaScript is arguably the most famous scripting programing language for HTML, allowing users to dynamically interact with the web application. The different architectures of WebRTC is discussed further. It is estimated that by 2019 more than 2million devices will be using WebRTC [14].

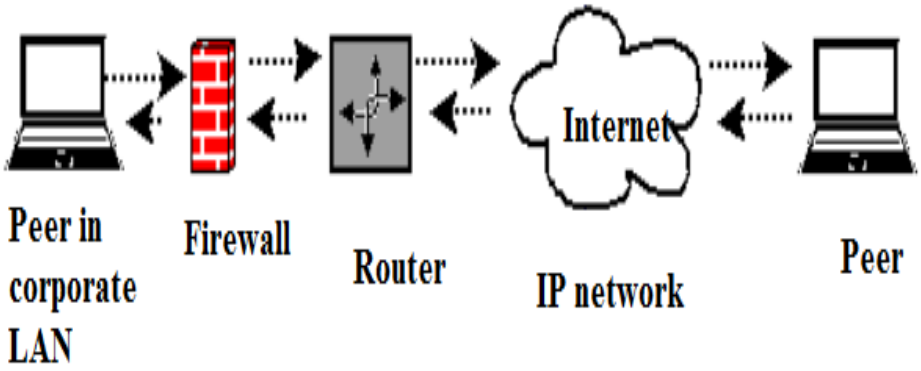

Figure 2.2: General Architecture of WebRTC

2.2.1 MCU (Multipoint Control Units)

Multipoint Control Unit usually converges all the video and audio streams into a single stream where all the clients are connected to single port. In MCU the server receives all the streams from the clients then it decodes and again encodes then into a single stream and sends back to the respective clients. This results in increase the load on the server with increase in the clients [15].

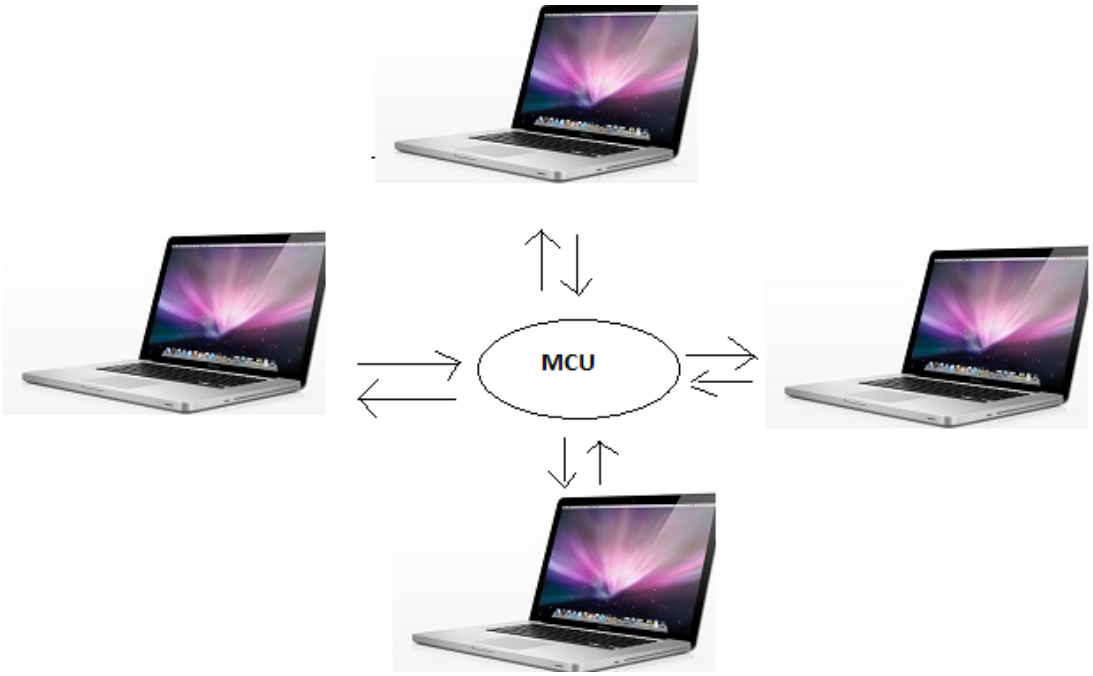

Figure 2.3: Multi Control Architecture

As the no of clients increases the complexity of the implementation also increases. So ultimately we get a high quality of video at the same time it costs a lot when compared to other architectures.

2.2.2 Selective Forwarding Unit

Selecting Forwarding Unit, In SFU all the clients send different video streams with different codecs and bitrates [16]. The server converts all those into a single entity and sends back that to all the connected clients. In this case clients will receive many video streams [17]. SFU uses vp8 codec for the media streams which is not supported by some browsers. SFU with simulcast has more benefits, in this case the clients send two media streams, one with high bit-rate and the other with low bit rate, so the server can choose the bit-rate according to the band width. By this the latency can be decreased.

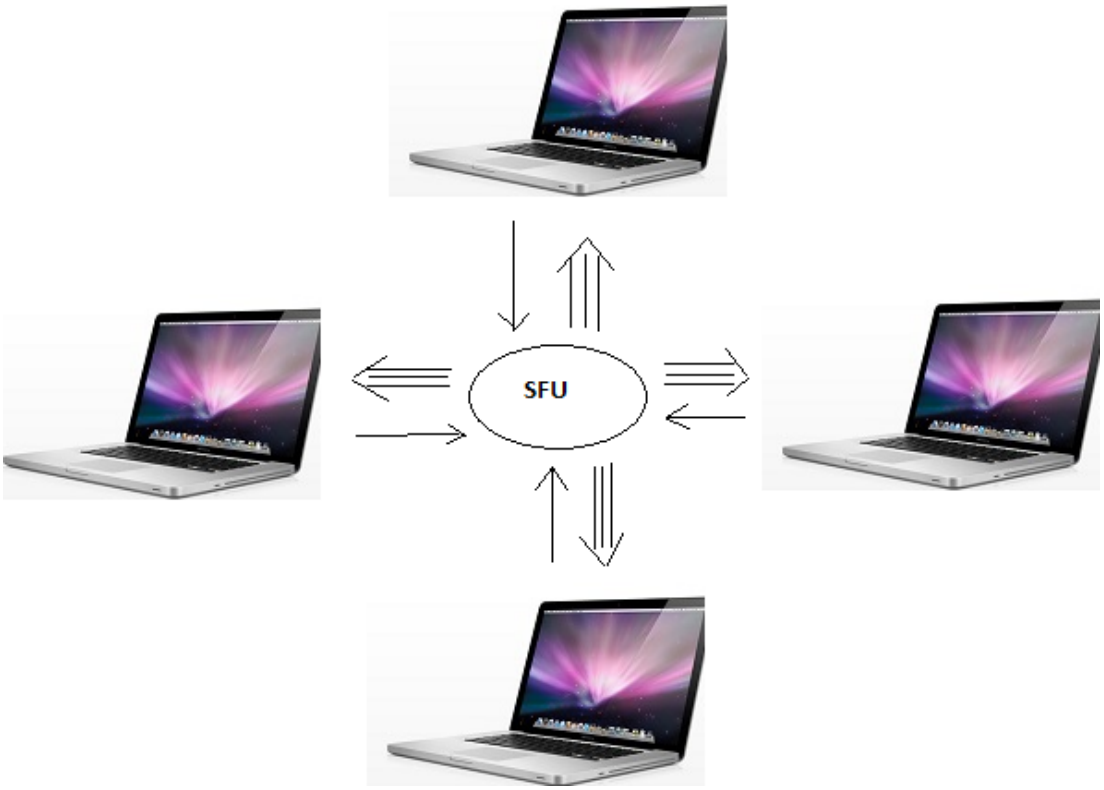

Figure 2.4: Selecting Foreword Unit Architecture

But on the other hand in SFU architecture the client receives multiple media streams, which results in consumption of more band width for downlink than uplink [16].

2.2.3 MESH

MESH architecture is one of the most widely used architecture in the WebRTC applications. This architecture is usually used for the small scale multimedia conferencing. In this architecture every individual client connects to other client and exchanges the data and media streams directly between the browsers without any involvement of server [18]. This saves a lot of cost on the server maintenance. This architecture based on the bandwidth can scale up to 10 remote participants.

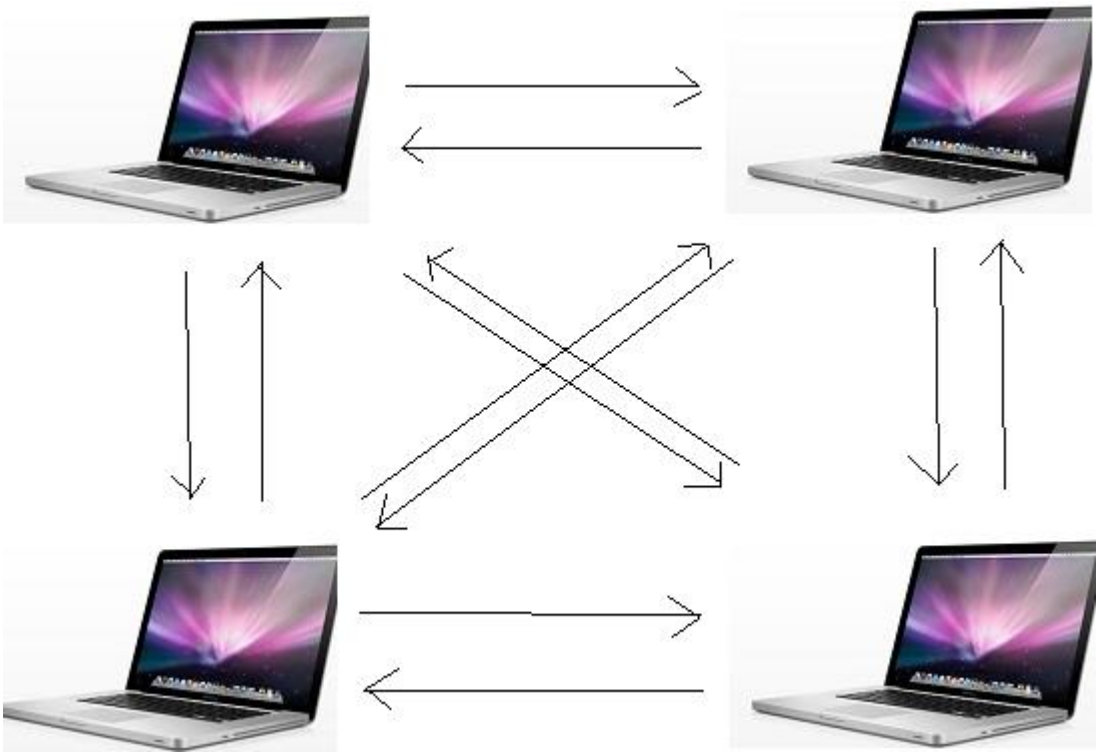

Figure 2.5: Mesh Architecture

As we can see that MESH architecture is the most suitable one when a user can establish a single end point connection. We see that in Mesh it is having a high node level degree, so that it enables low latency for the user.

2.3 WebRTC Components

The main advantage of WebRTC when compared to other frameworks is it includes Interactive Connection Establishment (ICE) [19][20].

2.3.1 Session Description Protocol

Session Description protocol (SDP) is a format for describing streaming media initialization parameters [21]. SDP is not responsible for delivering the media, but it describes the media format and type between the end points.

2.3.2 Media Stream

Media stream in WebRTC is mainly responsible to get the media streams from clients. The media can be in any form, if it is a microphone we get audio stream and from camera we get video. Media Stream is also liable for taking an input and output, input is the one taken from the camera and microphone, where output is responsible

for sending the media streams to the desired endpoints. Usually Media stream is generated from the browser using getusermedia(). getusermedia() is the API in media streams which prompts for the client permission to get access for the use of camera and microphone by the browser [22]. If the permission is granted by the client then getusermedia() will invoke all other clients with a value 1 which mean got a input of video, audio or shared screen, if the client denies the permission then it will invoke value 0 [23].

2.3.3 RTCPeerConnection

In WebRTC, RTCPeerConnection is responsible for the establishing a peer-to-peer connection between the browsers and also sets the connection from signaling state to stable state. WebRTC architecture uses mainly three different components such as audio, video and transport mechanism [24].

Main role of audio mechanism is to deal with echo cancellation and nose reduction. Audio component contains iSAC and iLBC codecs, iSAC codec is mainly responsible for the audio streams, this codec was developed by Global IP Solutions in 2011 and added to WebRTC for improving the quality of the audio stream in WebRTC.

Video mechanism is responsible for the video stream in the WebRTC, it consists of the VP8 codec. VP8 codec was developed by On2 technologies in 2008 and supported by all the modern browsers. It mainly helps to enhance image quality and audio/video buffer jitter.

Transport mechanism consists of SRTP (Secure Real Time Protocol) mainly responsible for the data streams in WebRTC.

There is a getstats() API for retrieving the statistics for the WebRTC on client side. This API helps the developers to analyze the statistics of their application. The structure of the API follows four stages [25] sender media capture statistics, sender RTP statistics, receiver RTP statistics, Receiver media statistics

2.3.4 RTCDataChannel

Data streams in WebRTC are exchanged in both directions from the clients. This exchange is done through RTCDataChannel mechanism. Using RTCDataChannel the delivery status of the message can be known. Text messaging and file sharing are done using RTCDatachannel. By this there would be low latency [26].

2.4 Storage Solutions for Real-Time Communication

Storage play's a major role in today's real time communication. In designing a storage for RTC, there are many factors to consider, performance scalability and cost [27]. In a conference meeting when some important files are shared between the remote sites. Sometimes chat messages, they should able to be archive even after the meeting. As there many different ways of using data bases for storage like, OrigoDB, Azure table storage etc. This thesis discusses about Azure Storage Table and OrigiDB.

2.4.1 Azure Storage Table

Azure storage is a product of Microsoft commonly used for hosting, deploying and managing applications. Azure storage Table is one of the brainiest solution for real-time applications. The storage accounts on azure run on a new flat network topology and fully supports scalability, without considering how they are created [28].

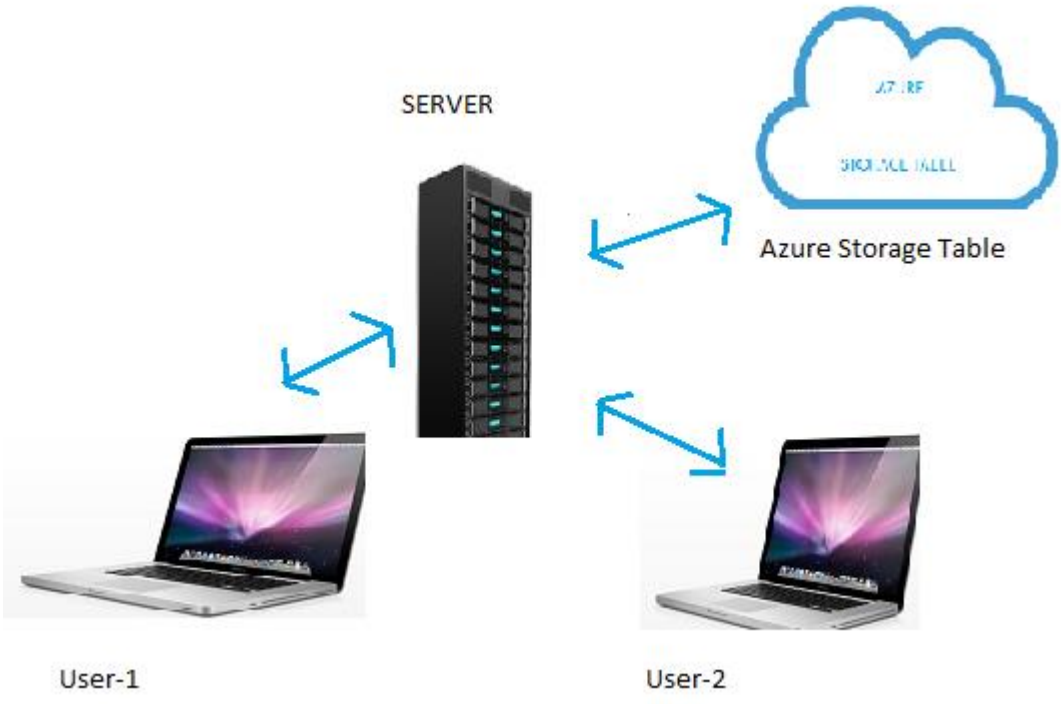

Figure 2.6: General setup for Azure Table Storage

From the figure we can see that when a user sends a data to the server, server will query that data to Azure Storage. The data is stored in the form of tables in the cloud. When a user tries to retrieve the data back, then server will again retrieves the data from the Azure Storage Table.

2.4.2 OrigoDB

OrigoDB is a data base which can be used for storing binary data in the real time applications. OrigoDB uses ram memory as the primary storage in the server [29].

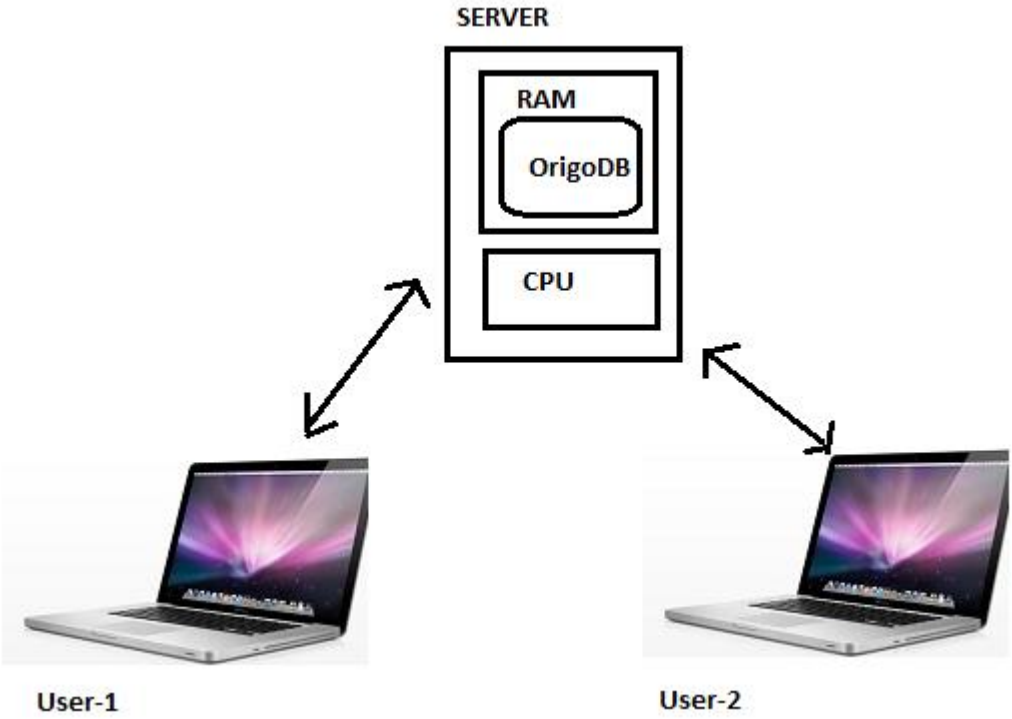

Figure 2.7: General setup for OrigoDB

OrigoDB helps developers to build an efficient data base, with a low cost and less time. Now-a-days the ram memory in the server is nearly reaching to 1TB, which means it can hold the 99% of the OLTP database memory.

2.4.2.1 Modular Architecture of OrigoDB

The main advantage of having a modular architecture is, it is very easy to integrate with any other application, modify the existing modules according to user needs. Easy to remove or add modules [29]. Most of the OrigoDB models are written in C#, so this makes very easy to create own models or to modify the existing models. OrigoDB server can be used in windows, Linux, mac, it is open source engine.

2.5 Bandwidth Utilization

The existing technologies which are used for real-time collaborations, WebEX, Blue jeans. These technologies don't have flexibility to adapt to network conditions. In

most cases when band width is low the users end up with having low quality of audio and video or having a frozen video. This may cause the interruption of the meeting. If a user is using some old configuration system, then there is possibility of frames getting held up in the browser.

Some of the techniques are observed in the user tests. When a user is not having a clear quality of media streams then usually they are advised to turn off their video. One of the most important requirements that we got from the observations is to have a better audio quality than a video. This scenario happens most of the time and the remote sites doesn't feel very comfortable when they don't have a video. It is very important to have a video in the complex decision making systems.

3 RELATED WORK

In [20] paper the authors explains the importance of the decentralized system. Even though the centralized systems here use a migration sever to connect the end points. Which ultimately results in usage of large network, is liable to the loss or when removing the key nodes and complicated when compared to decentralized system. This paper also proposed to solve the problems linked with node by setting up a peer-to-peer connection. Peer connection object is created to overcome the NAT problems using stun server so that the peer can locate the node which is outside the subnet.

When a client creates a connection request using create offer in WebRTC, then the information is combined by the STUN server with the local information provided by the client and sent to the server. If the server receives the same information from the other clients as well then the server will contact the client whose is offering the original signal, then the two clients can connect to each other. These two devices are identified and added to the network, so they can communicate directly without any server requirement. These clients can easily remove from the peer-to-peer network by using disconnect messages. By this the client's context Id is removed from the server. When a client tries to connect to network first the device will run a Migration Manager (MM) which is responsible for starting and receiving requests. This paper finally concludes than when MMs are integrated with all the migratable devices in our system, which are developed using the latest technology like HTML5, CSS3 and JavaScript.

Although this paper[20] explains the successful migration of the server form one host to another host, full architecture and effective usage of WebRTC but it fails to implement the full environment

In paper [30] sredojev describes about the implementation of the WebRTC technology and its Implementation of the client, server signaling. It mainly states that in WebRTC signaling methods are not specified to avoid redundancy and to increase the compatibility with the established technologies. This signaling process mainly uses the protocols such as SIP (Session Initiation Protocol), XMPP (Extensible Messaging and Presence Protocol), XHR (Xml Http Request) and SIP over WebSocket. In this paper for signaling it uses SIP over Websocket.. In WebSocket connection once the peer is connected to the server the connection is left opened until it is closed. The messages can be either text or binary it is exchanged between the peers and server.

Using of P-2-P systems also decreases the standard point and expenses [31]. It truly states that WebRTC is a cross-platform based application and plug-in free to use

in any type of device. In this paper the authors mainly focused on the performance evolution of WebRTC in mobile devices with respective of cost for establishing WebRTC connection, efficiency of WebRTC in multiple simultaneous data transfers, resource utilization caused by multiple simultaneous data transfers in WebRTC . The performance challenges in mobile devices are due to their hardware capabilities, wireless bandwidth and battery life [31]. Arto Heikkinen el did their performance evolution on WebRTC on mobile devices in terms of multiple simultaneous file transfers, session establishment delay and over-head. They authors did this on fully implemented WebRTC architecture with different mobile devices and different bandwidth and browsers combinations.

In paper [32], the technique of face detection and its advantages in real time communications was explained. Usually face detection techniques have dependencies like platform or windows operating system. This paper explains how to solve those type of issues by using java script API which are easily Integra table and can be used in real-time communications. This paper also evaluated two different face detection algorithms technique, CLM Constrained Local Model and CCV. Finally conclude by saying that face detection can play a key role in WebRTC. WebRTC also performed very well in terms of detecting face in various scenario's like bandwidth over Wifi and Lan, effectiveness of tracking etc. Finally this paper proposes that CLM algorithm is having most significance for detecting face in almost all light conditions.

In [33] paper, It is shown that the performance evolution of the of WebRTC over LTE is done. LTE is also a technology which helps the mobile phones to communicate over the mobile data (3G/4G). In this paper the performance evalution is done on the basis of the throughput, jitter, packet loss. This experiment is conducted in the NS-3 Environment for the LTE and WebRTC is hosted on a PC using Node.Js. This experimental procedure is done in 4 different scenarios. For the future work this paper proposed the scalability of the increase in number of Nodes.

In [27] paper various data bases for the real time communications has been explained. Mainly this article examined the impact on the under laying architecture on the performance of a distributed RTBDS [27]. It is also shown that how realistic should be an assumption of the Network delay to a data base. Two scenarios are considered for performance evolutions, CSMA and VTCSMA. The results in this paper show that, the performance of the real-time protocols doesn't need to show the better results in all the conditions. The performance improvement observed to be at a higher level compared to the improvement of the VTCSMA/CD over CSMA/CD [27, p. 016].

4 METHODOLOGY

Our main goal in this thesis is to develop a web-based collaboration tool which is not only good at performance but also to provide a solution according to the user requirements which are obtained from the user tests performed by User Interaction designers' Malin and Hedda in Semcon and Ericsson. The detailed experimental procedure is explained in the later sections.

4.1 Experimental Methodology

Initially we started research with the existing collaborative tools to see the available technologies and the architectures which are being used. Our aim in this project is to develop a web-based multimedia collaboration application which is scalable and having a secure architecture. This thesis mainly discuss about the Implementation of a suitable WebRTC architecture for an effective remote collaboration.

Secondly our aim is to choose a best WebRTC architecture which should be having a very low latency, high through put. For this a research was carried out for 2 weeks and has to choose one among Mesh, SFU and MCU. So based on the input and requirements of UI designers, we choose Mesh technology would be the most suitable one to use.

4.1.1 Framework for Signaling

As discussed in the back ground work, Mesh architecture is capable of scaling when there is increase in the no of participants. This Architecture also doesn't use much server resources as this directly communicates with the browser.

The frame work which should be used for WebRTC can be choosed purely on the basis of requirement. In this project we selected Node.js and Xsockets framework for the signaling mechanisms. Node.js is also a best architecture which is one among many. As Node.js is also web based technology built up on the chrome, which is very scalable for most of network applications. Node.js is light weight and efficient, suitable for many web applications.

XSockets Framework is mainly used for the signaling purpose, XSockets framework enable a real time communication in any device which is having TCP/IP and its architecture is very modular to add or replace the existing API's. XSockets

have a custom protocol which allows us to communicate with the clients in regardless of their protocol.

In this thesis we used XSockets framework for signaling mechanism between the clients. The selection on the framework is purely done on the basis of performance evaluation of XSockets over Node.js.

4.1.2 Bandwidth Adaptation

In WebRTC there is need of optimization of bandwidth. In this we focused on using face detection technique. Using this technique, no of participants in a remote site is analyzed and the media constrains like video resolution are been set to use accordingly. If a camera doesn't detect any face, then the transferring of the whole video feed throughout the session can take more bandwidth, when a client is using 3G/4G data to attend a remote meeting then this can consume a lot bandwidth.

The main reason of having this type of techniques is to enhance every possibility of improving scalability in Mesh architecture. As, we discussed in the background the Mesh architecture is having poor scalability. Using face detection technique the band width is kept under control, so that in case of additional of new remote sites in a meeting doesn't affect the system.

Here the media constraints are adjusted according to the number of faces detected. Here the minimum resolution used is 320x180 and maximum resolution used is 1280x720. The selection of the resolution can be done on the requirement of the application.

4.1.3 Selection of Database

According to the user requirements and feedback that we got from the UI designers, there is need for the data base to store the binary data. Our main goal is to select a data base which is having best response time and to save the resources corporate companies.

For this we chose OrigoDB as a data base for the storage of binary data. As discussed in the back ground it is very clear that OrigoDB is a best reliable data base. Though it uses ram memory for the storage, for our prototype we have a time period for storage. So that our prototype deletes the past data after that time. The evaluation of the data bases are done before selecting.

5 EXPERIMENTATION

In this thesis the performance of WebRTC is done in two experiments one is for evaluating signaling mechanism in WebRTC. The other experiment is for optimization of the WebRTC media streams through face detection technique. These experiments are evaluated based on the Network, cpu, memory and disk utilization.

5.1 Technical Specifications

Computers	**Server: Os-Win7 enterprise, cpu@2.40GHz, Ram-16Gb, Browser-.** **Client: Os-Win7 enterprise, cpu, Ram-,** **Browser-.**
Web Servers	Windows IIS-7 v. Apache v2.4.1.8 for windows which comes under XAMPP

5.2 Experimental Setup

5.2.1 Experiment-1

In this experiment for the selection of the framework that should be used upon the WebRTC is done on performing some evaluation tests on signaling mechanism.

As discussed earlier the mesh architecture in the WebRTC needs a signaling mechanism for the establishment of the P-2-P connection between browsers. Unfortunately, WebRTC is not a fully developed technology, so only few modern browsers support WebRTC. For this experiment two frameworks are choosed for signaling, XSockets and Node.js.

The experimental procedure is done within the same WiFi LAN. Here when a client A tries to establish a connection, then the localhost servers IIS and Apache tries to connect the Websocket server. Once the connection is established then WebRTC offers its SDP to the signaling server XSockets. The same procedure is done with client B. When Xsockets server receives the same SDP request then it establishes a connection between client A and client B. The experimental setup is shown in Figure

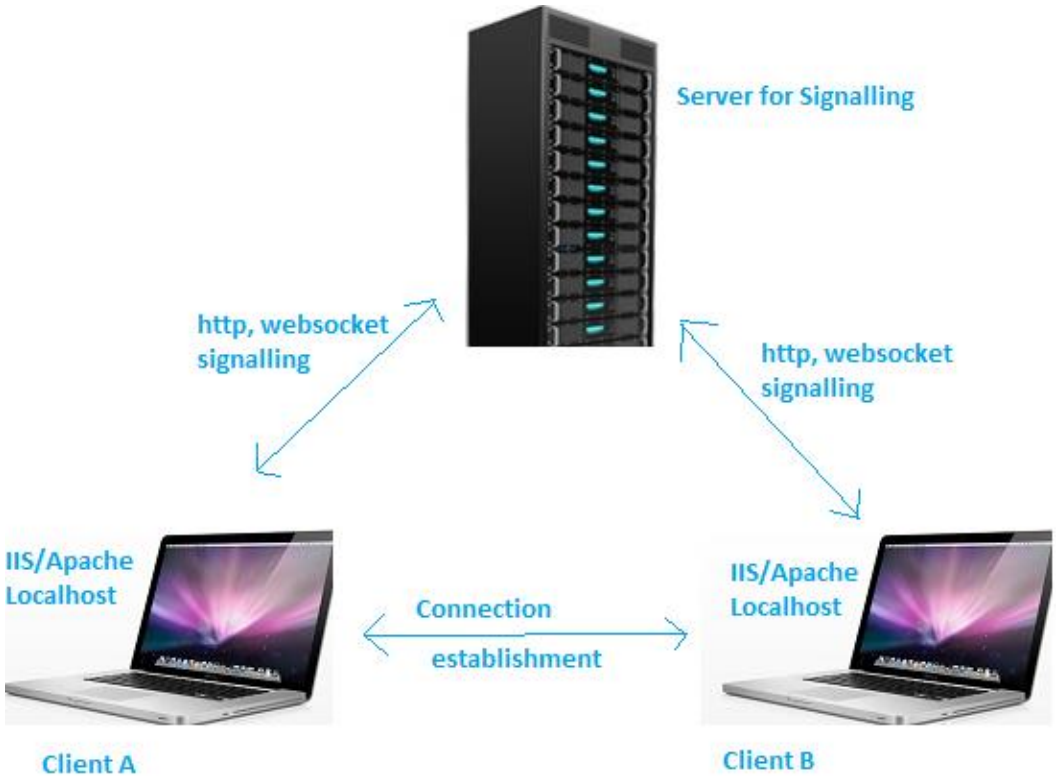

Figure 5.1: Architecture for Signaling Mechanism

This experiment is carried out in two different modern browsers (Chrome and Firefox). Connection Establishment between client A and client B are taken as call setup time. The number of connections is increased in the Client A system from 1 to 9. The variation of time is taken on client B. The same measure was repeated for 9 times and the average value is recorded. Same procedure is repeated with Apache server with Xsockets signaling server, Apache server with Node.Js server IIS server with Xsockets signaling server, IIS server with Node.js.

Parameters Used:

- Signaling Server: XSockets, Node.js.
- Localhost Server: IIS, Apache.
- Browsers: Chrome, Firefox.
- No of Clients: 1 to 9.

Assumptions and Regulations

- There are no other applications running on the local system and client system server and it is considered as only browsers are using complete network.

- Tried to maintain constant band width over LAN, for the whole experiment.

5.2.2 Experiment-2

The main of this experiment is to evaluate the performance and improve the scalability of the WebRTC mesh architecture. Face detection is used for constraining the media streams on the client side. If less number of faces is detected, then in media streams the video resolution can be adjusted accordingly. With this remote sites having with more number of clients can an opportunity of sharing a better quality of video.

In this experiment, face detection algorithm is performed on the client side, independent of the server and modifications are done in SDP protocol of WebRTC.

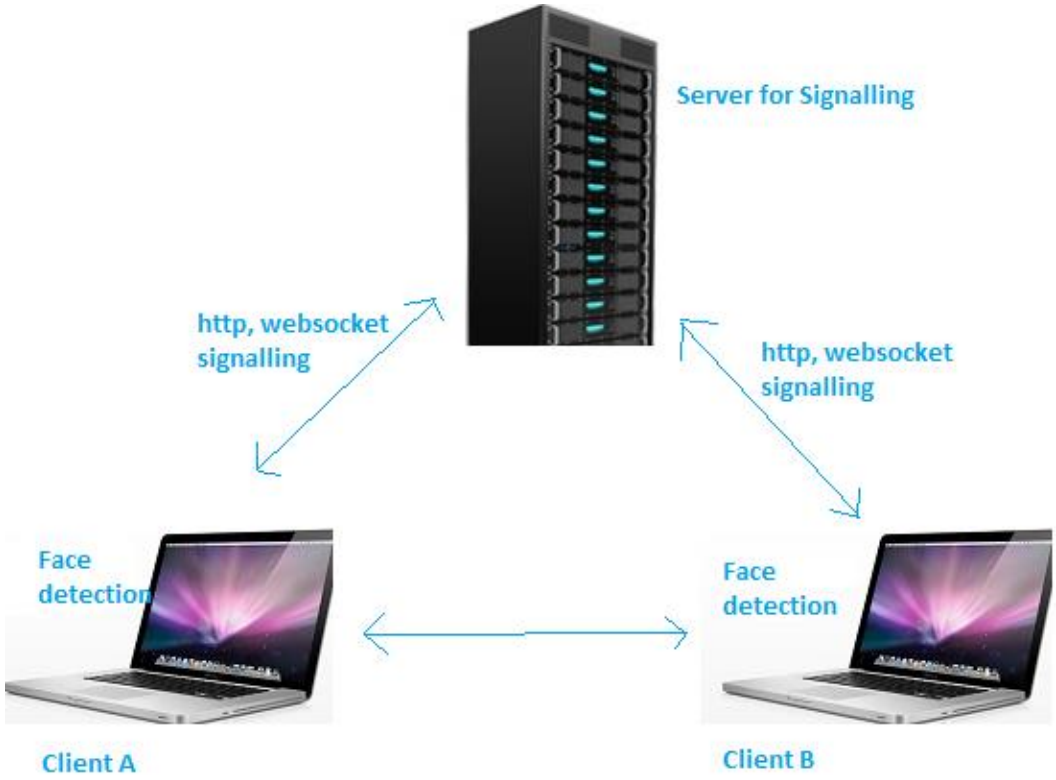

Figure 5.2: Experimental Setup for Face detection Technique

The above figure shows the architecture of the Experiment made. Here JavaScript library tracking.js is used for the detection of the faces. The network usage (CPU, Disk, Network and Memory) is also monitored in this experiment.

This experiment is carried out in two stages, One with the built-in configurations of the WebRTC and the other with the face detection. When number of more faces are detected then the video resolution is increased accordingly.

The performance of the system is measured with face detection and without face detection.

Parameters Used

- Signaling Server: XSockets, Node.js.
- Localhost Server: IIS, Apache.
- Browsers: Chrome, Firefox.
- Minimum number of participants-1.
- Maximum number of participants-6

Assumptions and Regulations

- There are no other applications running on the local system and client system server and it is considered as only browsers are using complete network.

Tried to maintain constant band width over LAN, for the whole experiment.

5.2.3 Experiment -3

In this experiment the primary focus is to evaluate the response time for the data bases. The comparison between data bases is done between origoDB and Azure Storage Table.

This experiment is carried out by testing the input response time and the output response time taken by the binary data for querying to the databases.

For these evolutions the origoDB and Azure Storage Table data bases are installed in the local machine. When a client a queried to the data base the input time response is calculated and the then the same data is retrieved and the out response time for this is also calculated. These experiments are performed in the standard browsers like Chrome and Firefox.

The input response time and output response time for the data that is queried. JavaScript function is return on the client to calculate the input and output response. Time stamps are used to calculate the Reponses when they are created. For getting the time stamps, a simple JavaScript function is added in the client side. The Reponses are queried for 10 times for the input and output and the average values are taken. The results are shown in Results section.

6 RESULTS

This results section is divided into three parts

6.1- Results of Signaling Mechanism.

6.2- performance evolution of the WebRTC with respective of face detection.

6.3-Input and Output response time of databases in RTC.

6.1 Results of Signaling Mechanism

Experimental results obtained from the signaling servers XSockets and Node.Js

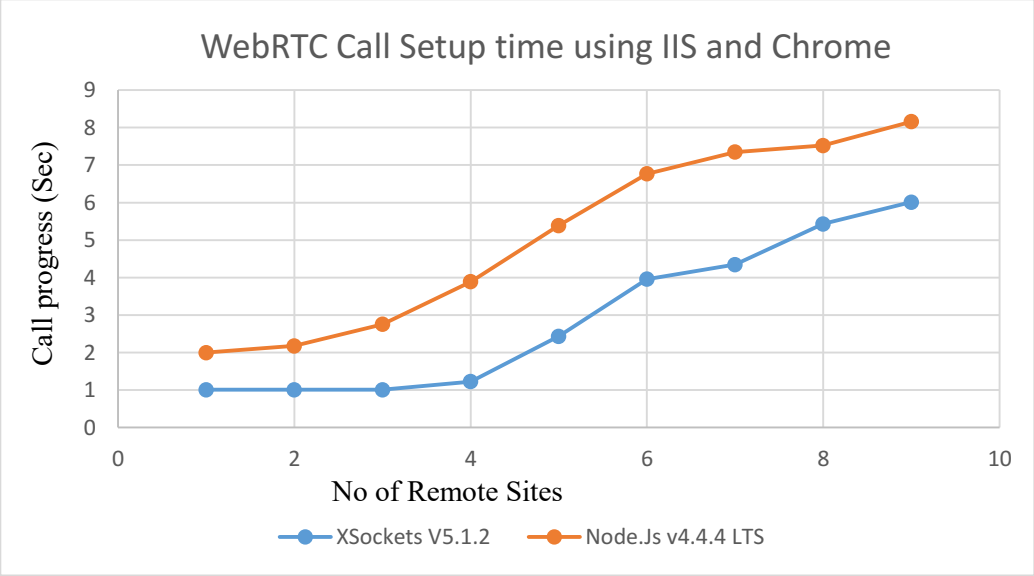

Figure 6.1: Call setup time using IIS and Chrome

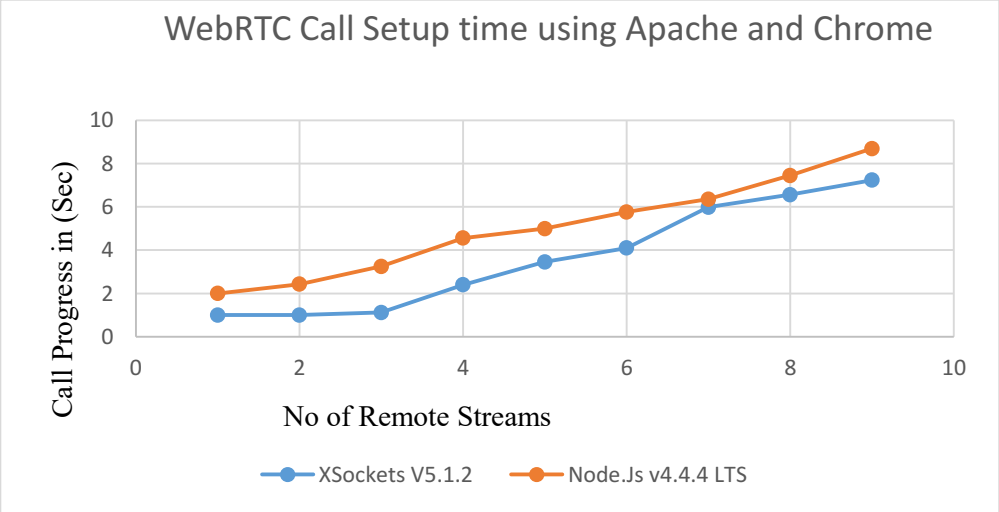

Figure 6.2: Call setup time using Apache and Chrome

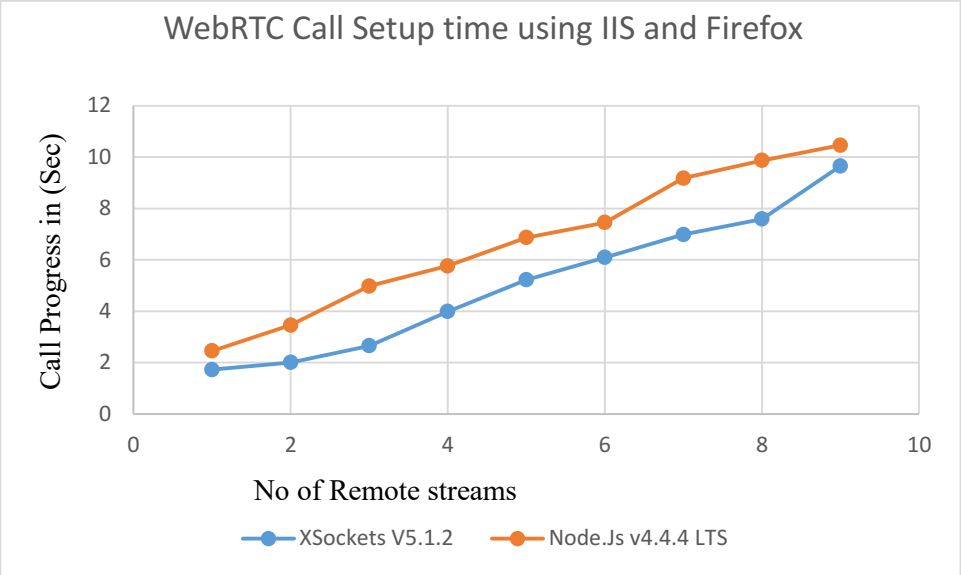

Figure 6.3: Call setup time using IIS and Firefox

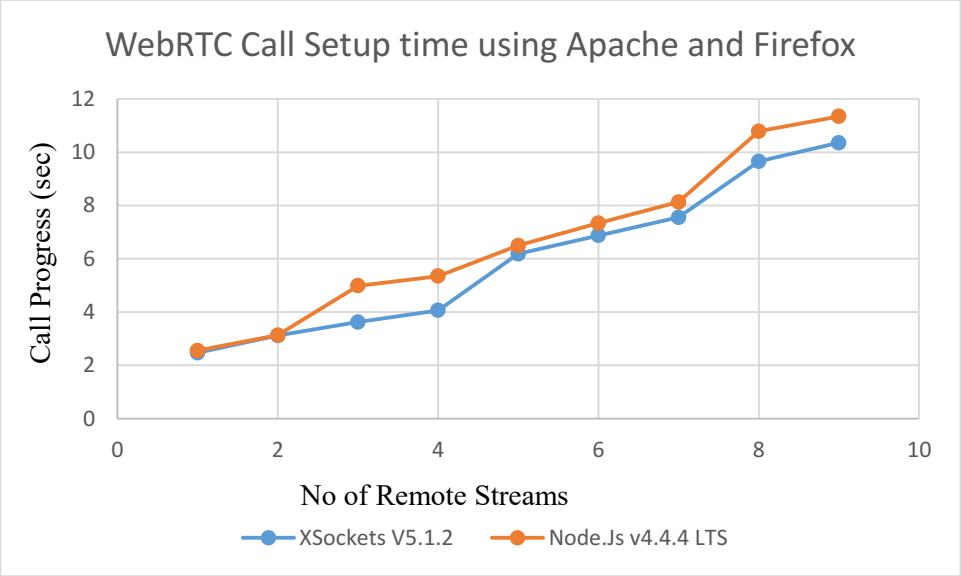

Figure 6.4: Call setup time using Apache and Firefox

From the above figures 6.1, 6.2, 6.3, 6.4 experimental results it proven that XSockets Server is having low call setup time When compared to Node.Js server in different browsers Using Apache and IIS web servers. These experiments are performed on different tabs when 9 remote users try to connect. From the results it can be seen that although the graphs are linear, considering the time taken to setup a call by XSockets signaling server is low even when using different browsers and web server.

6.2 Performance evolution of the WebRTC with respective of face detection

From this experiment it is clearly proven that, with use of Face detection technique the resource utilization of a system and bandwidth utilization can be optimized.

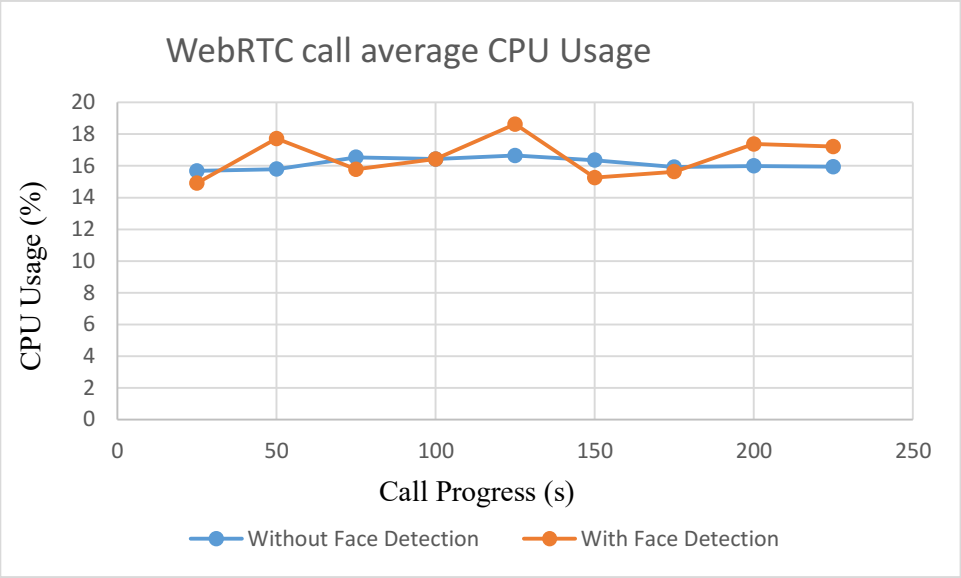

Figure 6.5: Average CPU usage of a WebRTC call with and without face detection

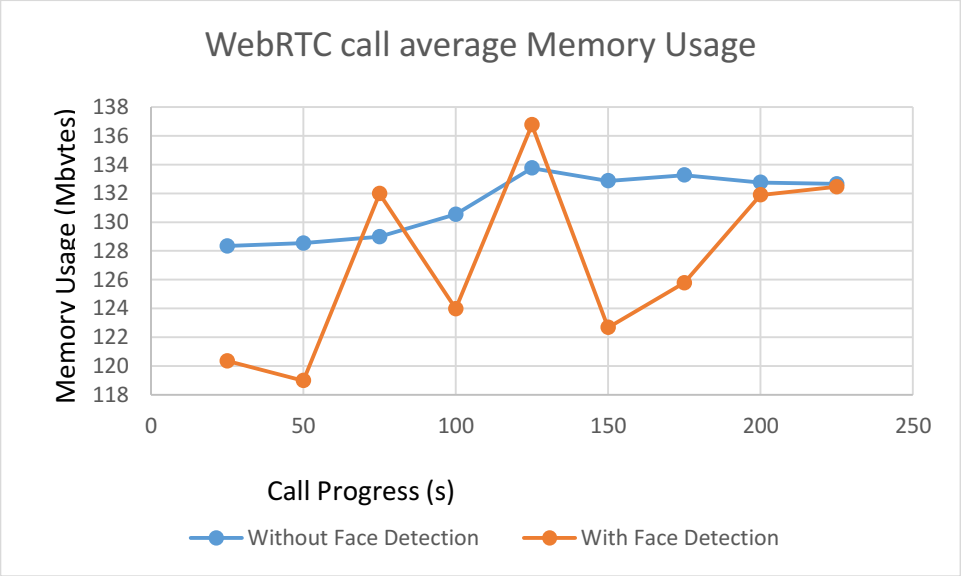

Figure 6.6: Average Memory usage of a WebRTC call with and without face detection

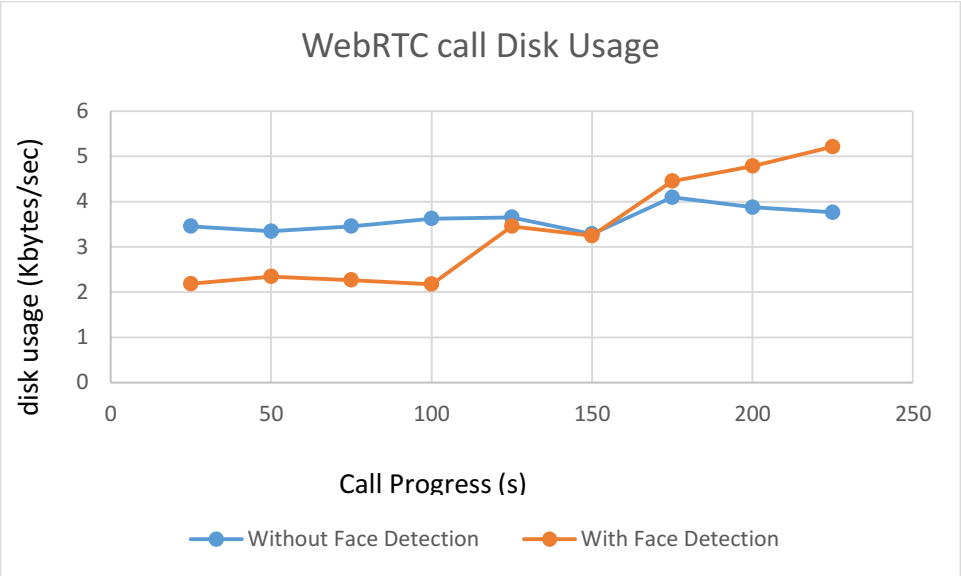

Figure 6.7: Average Disk usage of a WebRTC call with and without face detection

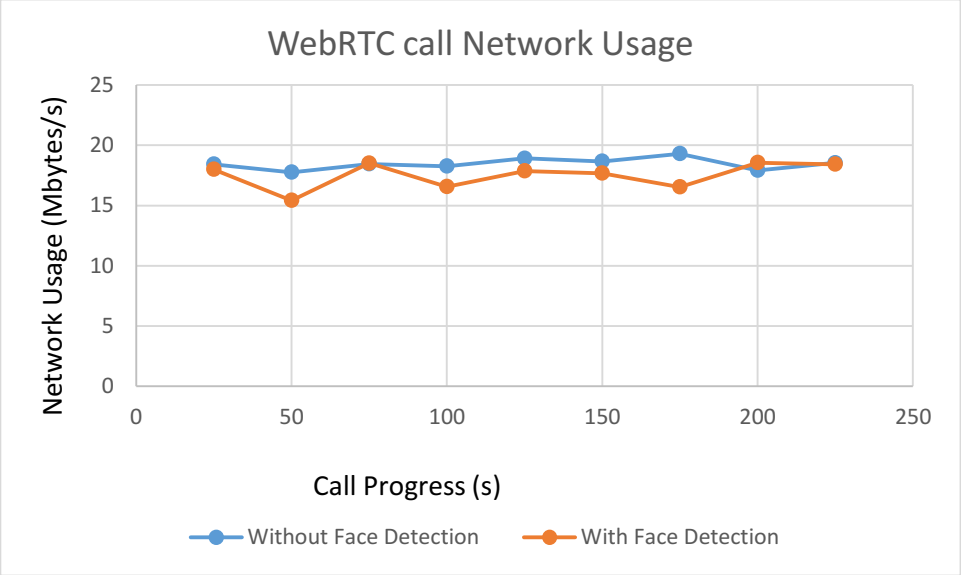

Figure 6.8: Average Bandwidth usage of a WebRTC call with and without face detection.

From the above experimental results we can see that there is dynamical change in the CPU, Memory, Disk and Network usage. This is because whenever a system detects the more number of faces the resolution of the video is increased accordingly. With this there is a need of using high bandwidth and system recourses for the high resolution video which also results in the increase of system recourses. Whenever there is less number of faces detected there is no need of sending high resolution so ultimately saves resources. The above experiment is carried in a call progress with duration of 225 (sec) using face detection and without face detection.

6.3 Input and Output response time of databases in RTC

In this experiment, Input and Output response of OrigoDB and Azure Table Storage. Results show that the response time of OrigoDB depends on the size of the file.

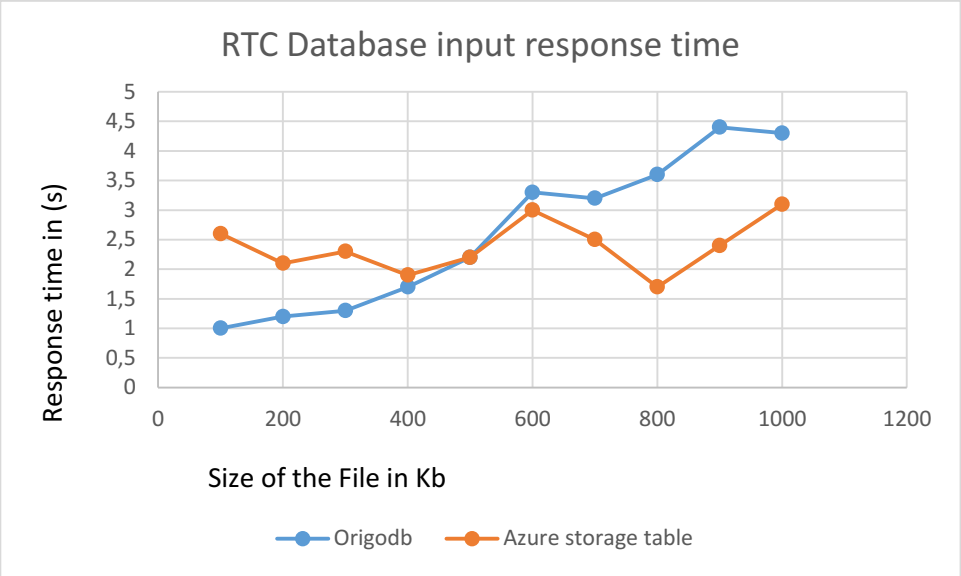

Figure 6.9: RTC input response time of a Data base

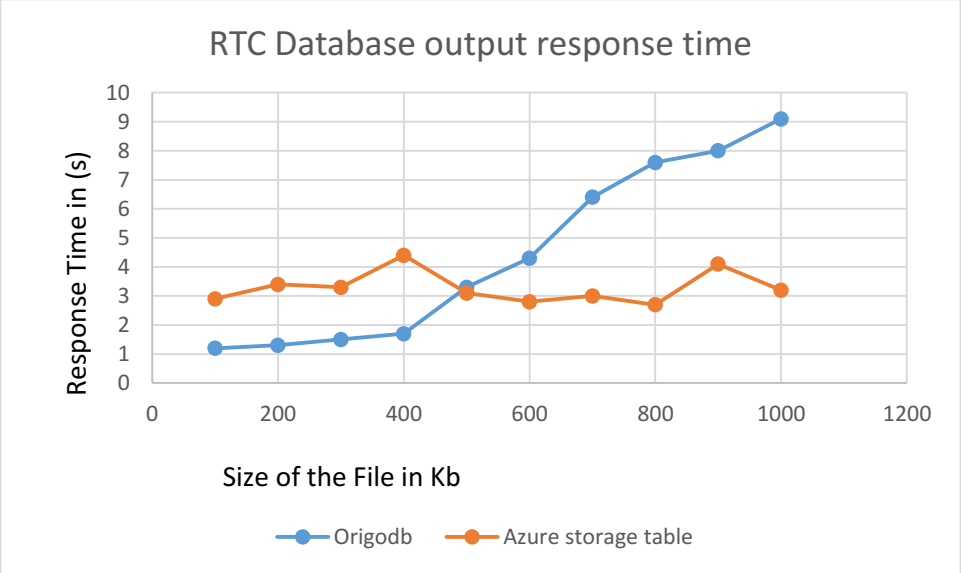

Figure 6.10: RTC output response time of a Data base

The above Figures 6.9 and 6.10 are the results obtained by the response time taken by the Azure table storage and Origo.db databases. Form the results it is clear that when the amount of data is increasing Origo.db database is also taking a bit longer. Finally I can say that the selection of the database purely depends on the developer needs to his application. Using Origo.db saves lot of recourses to the developers as it uses ram memory. If the file size is 500kb at once then Origo.db doesn't show any effect on the response time, as the file size increases the response time also increases.

7 ANALYSIS AND DISCUSSION

The main aim of this thesis is to find an appropriate architecture for the WebRTC which helps to improve the flexibility and performance of the WebRTC architecture. After a deep investigation, it is found that mesh architecture would be the most suitable architecture for the requirements of MERCO project. In terms of the resource utilization, development and maintenance, this architecture fits best as it doesn't require any server or cloud storage.

However, mesh architecture requires a signaling server for connecting with other clients. This is the trickiest part in whole project and also Crucial stage for building an architecture for MERCO project. For this purpose, two signaling server is chosen, XSockets and Node.Js. Performance evolution is made based upon the core call setup time taken between two clients. For this evaluation two webservers are IIS and Apache are considered. Modern browsers Chrome and Firefox are used in experiments. The experimental results show that XSockets shows the better results when compared to Node.js. After deploying the solution to the hosting servers, Xsockets found to be most reliable and efficient.

Later on one of the major issue that we recognized in this project was, present WebRTC SDP protocol is designed in a way that the whole bandwidth of the system is shared between the clients of a conference system, as it is under development. This mainly effects the scalability of and performance of a system. So, there must be some technique for constraining the media streams. For this purpose, Face detection technique is proposed and evaluated. For detecting faces tracking.js is used. When a minimum number of faces are detected the video resolution is reduced to minimum (320x180), when maximum number of faces detected resolution is set to maximum (1280x720). Experiment-2 shows that by using this technique the resource utilization of the browser can also be reduced.

For this project there is a requirement of the data base for storing the shared files and and chat messages. This issue was solved by choosing OrigoDB database. The selection of the database is done on the evolution of two databases OrigoDB and Azure Storage Table based on input and out responses of the databases. Even though from the results Azure Storage exhibited constant time stamps, it requires additional resources like subscription from Azure, which results in very poor scalability. As OrigoDB is installed in the webapp server, it shows very low latency and doesn't require any additional costs. So OrigoDB is choosed as the database for MERCO project, as it thoroughly meets the requirements.

7.1 Proposed Final Solution

The final solution of the MERCO project is based on the 5 thesis students, which designs are proposed by Mallin and Hedda (UI Designers), communication concept was developed by Artem (Communication Engineering in GU) and Penzi (front end developer). The final solution of MERCO project includes video/audio systems for conferences, public and private messaging, and queue system for participants, polling, and screen sharing and countdown timer.

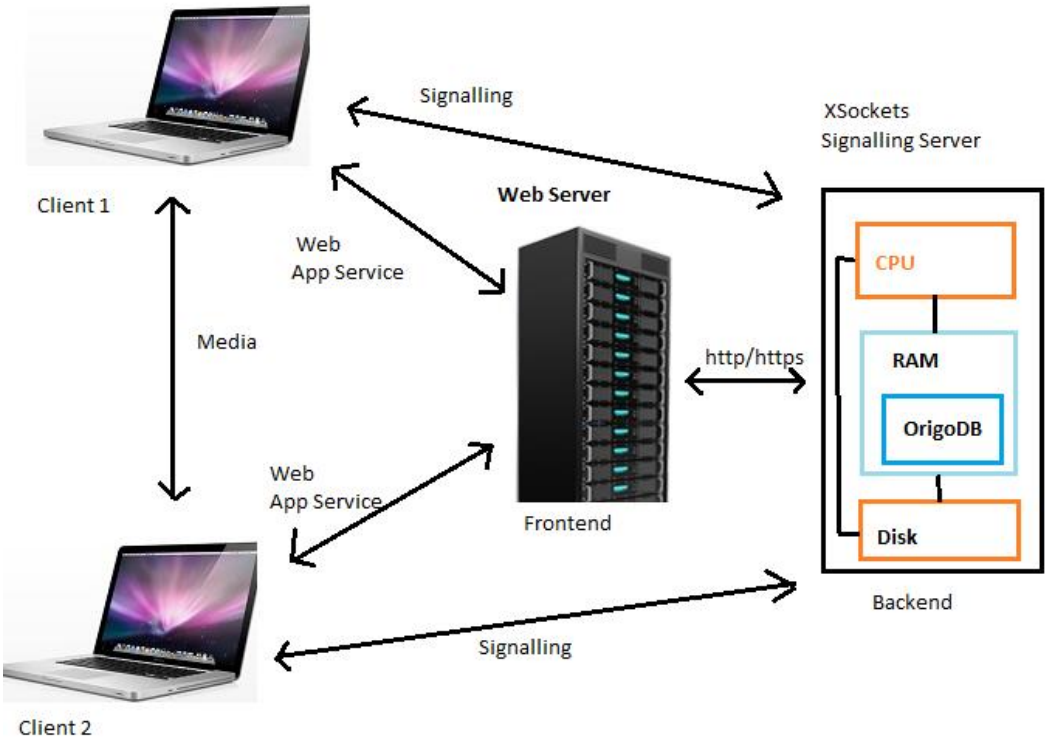

Figure 7.1: Final Architecture

Frontend

The Frontend is the face of the system; it is the one that interacts with the user. Technically, it is the web application which gives the WebRTC service to the user agent (browser). The technologies and programing languages used in the frontend end include HTML, JavaScript and CSS. The functions of the frontend are categorized into three stages, the lobby (invitation) page, the control (command) page and the video feeds page. The codes for the Frontend are deployed in the webserver and runs on the client in the client side web browser. The lobby page allows the user to setup a conference using a unique 128 bits context id. It also generates two urls (links) that will be used by other remote participants to join the

conference using either video or no video depending on the nature of device and connection environment. The lobby page also invokes the mail API to conference invitation through email. User can also set the meeting date, meeting start time and meeting end time and the agenda of the meeting. These meeting details will be send to the backend server and later used during the meeting. The command page carries a lot functions which comprise of the public and private chat, recording and tagging of session, session muting, screen sharing, raise question, 55 polling and agenda editing. It also communicates with the backend server for updating the stored parameters. The command page is the one carrying a lot of functions used in remote collaboration. The command page of a specific conference is opened by simple click on the link containing the unique meeting context id. The command page with video option automatically invokes the video feeds page. The command page also allows users to register the name of their remote site like company name or group name. The video feeds page is responsible for capturing and displaying feeds. This is where the WebRTC GetUserMedia API runs, it access the device camera and mic and prepares media to send to other clients. It also receives and displays videos and shared screens from remote sites. It detects if the user is speaking and adjust the media constraints to enable system flexibility of using less powerful devices. It also detects if the remote user is speaking and place adjust the speaking remote video size and position for better visualization. During the conference, the video feeds page also displays a countdown clock, the agenda and the conference progress bar. It also displays the registered company name or group names linked to their corresponding videos.

Backend

The Backend component is invisible to the user, its function are transparent. The Backend server is deployed in cloud and it must always keep on running to ensure service availability. It is the one which enable the conference participants to find each other. The technologies and programing languages used in the frontend end include XSockets, C#, Origodb. The backend server is made of XSockets controllers and the in memory Origo database all build in C#. The XSockets controllers are responsible for signaling and for passing messages and commands among the conference participants. The XSockets controllers are configured for a persistent storage property linked to the azure storage classic account using a Storage Connection String. The Origodb is an in memory database for storing instant chat messages and shared files during conferencing. It has very low response time for storing and retrieving data of around 500kB or less. The data is stored in RAM of the server and can be quickly retrieved by the joining participants.

8 CONCLUSION AND FUTURE WORK

The main objective of this thesis is to show the different architectures of WebRTC and their mechanisms. So that it helps developers to choose appropriate that suits their application in the initial stage. As WebRTC is still in developing stage and supported by only modern browsers, in future it opens many doors and possibilities.

Different WebRTC architectures and their uses were described in this thesis. Based on the requirements of the MERCO project Mesh architecture is used. Furthermore for signaling server, performance of XSockets and Node.Js are evaluated using different browsers and webservers based on total call setup time. Form the results obtained it is shown that XSockets is having less call setup time with different browsers and webservers. So, for MERCO project XSockets signaling server is used.

For the band width adaptation and resource utilization in MERCO project face detection technique is used. The test results obtained using face detection shows very promising for real-time communications. By using this technique, usage of band width by media constraints can be constrained. Which results in band width consumption reduction as well the Network utilization is optimized. By this the scalability of the mesh architecture also can be improved, as the application uses less bandwidth and resources.

The database response time is very important in the real-time communications. In this thesis the OrigoDB and Azure Storage Table, input and output response time are evaluated. As the results show that when the volume of data increases the response time of the Origodb is also increasing as it stores in RAM memory. This is happening only when the large data flies are shared in a meeting. But for the Azure Storage Table it is nearly constant. But in MERCO project it is assumed that only 2-3 files which is 200kb to 300kb size of files are only shared. For this purpose, OrigoDB works fast and more cost-effective.
Finally, I hereby conclude that, the final prototype of MERCO project is successful proof of concept of supporting Remote Collaborations with a better user experience, usability and conference scalability. The algorithms used also optimizes the resourse utilization.

As this is mainly focused on personal computers, future research could be done on integrating the APIs and WebRTC architectures for mobile versions and tablets, in

accordance with the resource and battery consumption. Considering security issues an own encryption method can also be implemented in WebRTC which should be supported by modern browsers. Though WebRTC is supported by many browsers, but there are problems during cross browser communications. These issues weren't addressed in this research which this can be solved further.

REFERENCES

[1] B. Bhargava, "Adaptable software for communications in video conferencing," in *1998 IEEE Workshop on Application-Specific Software Engineering Technology, 1998. ASSET-98. Proceedings*, 1998, pp. 8–13.

[2] "WebRTC Home | WebRTC." [Online]. Available: https://webrtc.org/. [Accessed: 14-May-2016].

[3] M. Mody, P. Swami, and P. Shastry, "Ultra-low latency video codec for video conferencing," in *2014 IEEE International Conference on Electronics, Computing and Communication Technologies (IEEE CONECCT)*, 2014, pp. 1–5.

[4] M. Adeyeye, I. Makitla, and T. Fogwill, "Determining the signalling overhead of two common WebRTC methods: JSON via XMLHttpRequest and SIP over WebSocket," in *AFRICON, 2013*, 2013, pp. 1–5.

[5] S. D. P. J. 23rd, 2012 Updated: February 21st, and 2014 Comments: 97 Your browser may not support the functionality in this article, "Getting Started with WebRTC - HTML5 Rocks," *HTML5 Rocks - A resource for open web HTML5 developers*. [Online]. Available: http://www.html5rocks.com/en/tutorials/webrtc/basics/. [Accessed: 14-May-2016].

[6] R. Li, J. Curhan, and M. E. Hoque, "Predicting video-conferencing conversation outcomes based on modeling facial expression synchronization," in *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, 2015, vol. 1, pp. 1–6.

[7] "Kurento." [Online]. Available: https://www.kurento.org/. [Accessed: 14-May-2016].

[8] "Introduction | XSockets.NET 5." [Online]. Available: https://uffebjorklund.gitbooks.io/xsockets-net-5/content/. [Accessed: 14-May-2016].

[9] T. Sandholm, B. Magnusson, and B. A. Johnsson, "An On-Demand WebRTC and IoT Device Tunneling Service for Hospitals," in *2014 International Conference on Future Internet of Things and Cloud (FiCloud)*, 2014, pp. 53–60.

[10] M. H. Hajiesmaili, L. T. Mak, Z. Wang, C. Wu, M. Chen, and A. Khonsari, "Cost-Effective Low-Delay Cloud Video Conferencing," in *2015 IEEE 35th International Conference on Distributed Computing Systems (ICDCS)*, 2015, pp. 103–112.

[11] V. A. P, S. Hari, P. K. P, V. K. J, and A. K. R, "Telemedicine for emergency care management using WebRTC," in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2015, pp. 1741–1745.

[12] N. Hongo, H. Yamamoto, and K. Yamazaki, "Web shopping support system for elderly people using WebRTC," in *16th International Conference on Advanced Communication Technology*, 2014, pp. 934–940.

[13] "real time communication - Google Search." [Online]. Available: https://www.google.se/search?q=real+time+communication&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiJrenF1uLMAhULGZoKHTzVBV4Q_AUIBygB&biw=1608&bih=820&dpr=0.9#imgrc=RMHa8SIT-8tG6M%3A. [Accessed: 18-May-2016].

[14] P. Chainho, K. Haensge, S. Druesedow, and M. Maruschke, "Signalling-On-the-fly: SigOfly," in *2015 18th International Conference on Intelligence in Next Generation Networks (ICIN)*, 2015, pp. 1–8.

[15] "319-W002.pdf." [Online]. Available: http://www.ijfcc.org/papers/319-W002.pdf. [Accessed: 14-May-2016].

[16] "WebRTC Multiparty Video Alternatives, and Why SFU is the Winning Model • BlogGeek.me," *BlogGeek.me*, 07-Mar-2016. [Online]. Available: https://bloggeek.me/webrtc-multiparty-video-alternatives/. [Accessed: 16-May-2016].

[17] "What are the Challenges of DIY your WebRTC SFU? • BlogGeek.me," *BlogGeek.me*, 25-Jan-2016. [Online]. Available: https://bloggeek.me/webrtc-sfu-challenges/. [Accessed: 16-May-2016].

[18] "Mesh," *WebRTC Glossary*, 08-Sep-2014. [Online]. Available: https://webrtcglossary.com/mesh/. [Accessed: 16-May-2016].

[19] H. Arslan, S. Tüncel, and A. G. Yüksek, “Comparison of the Web based multimedia protocols for NAT traversal performance,” in *2015 23nd Signal Processing and Communications Applications Conference (SIU)*, 2015, pp. 915–918.
[20] D. Johansson and M. Holmgren, “Towards Implementing Web-Based Adaptive Application Mobility Using Web Real-Time Communications,” in *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2014, pp. 483–486.
[21] “Session Description Protocol,” *Wikipedia, the free encyclopedia*. 22-Mar-2016.
[22] S. D. P. J. 23rd, 2012 Updated: February 21st, and 2014 Comments: 97 Your browser may not support the functionality in this article, “Getting Started with WebRTC - HTML5 Rocks,” *HTML5 Rocks - A resource for open web HTML5 developers*. [Online]. Available: http://www.html5rocks.com/en/tutorials/webrtc/basics/. [Accessed: 16-May-2016].
[23] “MediaStream API,” *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API. [Accessed: 16-May-2016].
[24] “RTCPeerConnection,” *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection. [Accessed: 16-May-2016].
[25] L. L. Singh Varun, “Basics of WebRTC getStats() API — callstats.io.” [Online]. Available: http://www.callstats.io/2015/07/06/basics-webrtc-getstats-api/. [Accessed: 16-May-2016].
[26] D. R. P. F. 4th, 2014 Updated: February 4th, and 2014 Comments: 2 Your browser may not support the functionality in this article, “WebRTC data channels: WebRTC data channels for high performance data exchange - HTML5 Rocks,” *HTML5 Rocks - A resource for open web HTML5 developers*. [Online]. Available: http://www.html5rocks.com/en/tutorials/webrtc/datachannels/. [Accessed: 16-May-2016].
[27] “PII: S0164-1212(96)00043-X - jss97.pdf.” [Online]. Available: http://www.cs.bilkent.edu.tr/~oulusoy/jss97.pdf. [Accessed: 15-May-2016].
[28] “Azure Storage Table Design Guide | Microsoft Azure.” [Online]. Available: https://azure.microsoft.com/en-us/documentation/articles/storage-table-design-guide/. [Accessed: 16-May-2016].
[29] “Start - OrigoDB.” [Online]. Available: http://origodb.com/. [Accessed: 16-May-2016].
[30] B. Sredojev, D. Samardzija, and D. Posarac, “WebRTC technology overview and signaling solution design and implementation,” in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015, pp. 1006–1009.
[31] A. Heikkinen, T. Koskela, and M. Ylianttila, “Performance evaluation of distributed data delivery on mobile devices using WebRTC,” in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2015, pp. 1036–1042.
[32] M. Phankokkruad and P. Jaturawat, “An evaluation of technical study and performance for real-time face detection using Web Real-Time Communication,” in *2015 International Conference on Computer, Communications, and Control Technology (I4CT)*, 2015, pp. 162–166.
[33] G. Carullo, M. Tambasco, M. D. Mauro, and M. Longo, “A performance evaluation of WebRTC over LTE,” in *2016 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, 2016, pp. 1–6.

APENDIX

Table 1: Call setup time using IIS and Chrome

No of remote Streams	XSockets V5.1.2	Node.Js v4.4.4 LTS
1	1.009	1.997
2	1.009	2.178
3	1.010	2.756
4	1.226	3.890
5	2.436	5.386
6	3.958	6.765
7	4.347	7.345
8	5.432	7.523
9	6.010	8.158

Table 2: Call setup time using Apache and Chrome

No of remote Streams	XSockets V5.1.2	Node.Js v4.4.4 LTS
1	1.000	1.996
2	1.004	2.428
3	1.120	3.248
4	2.398	4.560
5	3.456	4.990
6	4.096	5.764
7	5.987	6.356
8	6.564	7.448
9	7.236	8.690

Table 3: Call setup time using IIS and Firefox

No of remote Streams	XSockets V5.1.2	Node.Js v4.4.4 LTS
1	1.726	2.460
2	2.010	3.459
3	2.654	4.980
4	3.987	5.765
5	5.224	6.870
6	6.093	7.450
7	6.987	9.180
8	7.590	9.870
9	9.650	10.458

Table 4: Call setup time using Apache and Firefox

No of remote Streams	XSockets V5.1.2	Node.Js v4.4.4 LTS
1	2.470	2.555
2	3.120	3.127
3	3.625	4.986
4	4.063	5.348
5	6.184	6.497
6	6.876	7.338
7	7.554	8.126
8	9.656	10.786
9	10.349	11.346

Table 5: Average CPU usage of a WebRTC call with and without face detection

Conference call progress in (sec)	Without Face Detection	With Face Detection
25	15.68	14.91
50	15.79	17.72
75	16.54	15.78
100	16.43	16.43
125	16.65	18.63
150	16.36	15.26
175	15.92	15.63
200	15.99	17.38
225	15.94	17.22

Table 6: Average Memory usage of a WebRTC call with and without face detection

Conference call progress in (sec)	Without Face Detection	With Face Detection
25	128.34	120.35
50	128.54	118.98
75	128.98	131.98
100	130.54	123.97
125	133.76	136.76
150	132.87	122.67
175	133.27	125.78
200	132.76	131.89
225	132.65	132.45

Table 7: Average Disk usage of a WebRTC call with and without face detection

Conference call progress in (sec)	Without Face Detection	With Face Detection
25	3.457	2.184
50	3.346	2.346
75	3.457	2.267
100	3.627	2.176
125	3.654	3.456
150	3.286	3.246
175	4.096	4.453
200	3.876	4.786
225	3.765	5.213

Table 8:Average Network usage of a WebRTC call with and without face detection

Conference call progress in (sec)	Without Face Detection	With Face Detection
25	18.42	18.00
50	17.76	15.42
75	18.45	18.52
100	18.26	16.56
125	18.92	17.87
150	18.67	17.68
175	19.30	16.54
200	17.92	18.56
225	18.54	18.42

Table 9: RTC input response time of a Data base

Data size(Kbytes)	Origodb	Azure storage table
100	1.0	2.6
200	1.2	2.1
300	1.3	2.3
400	1.7	1.9
500	2.2	2.2
600	3.3	3.0
700	3.2	2.5
800	3.6	1.7
900	4.4	2.4
1000	4.3	3.1

Table 10: RTC output response time of a Data base

Data size(Mbytes)	Origodb	Azure storage table
100	1.2	2.9
200	1.3	3.4
300	1.5	3.3
400	1.7	4.4
500	3.3	3.1
600	4.3	2.8
700	6.4	3.0
800	7.6	2.7
900	8.0	4.1
1000	9.1	3.2