# **REAL TIME** **SECURED COMMUNICATION SYSTEM USING WEBRTC**

**Peter Chidubem Atunalu - 144013**

Submitted to the

Graduation Projects Examination Jury

in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

European University of Lefke

May 2018

Approval of the Computer Engineering Department

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Assoc. Prof. Dr. Hüseyin Ademgil

Chairman/Director

This is to certify that we have read this graduation project and that in our opinion it is fully adequate, in cope and quality, as an Undergraduate Project.

Assist. Prof. Dr. Zefer Ernel

Supervisor

Members of the examining committee

Name Signature

1. Assoc. Prof. Dr. Hüseyin Ademgil …….………………………

2. Assist. Prof. Dr. Cem B. Kalyoncu …….………………………

3. Assist. Prof. Dr. Ferhun Yorgancıoğlu …….………………………

4. Assist. Prof. Dr. Vesile Evrim …….………………………

5. Dr. Ersin Çağlar …….………………………

Date: …….………………………

# Abstract

**REAL TIME SECURED COMMUNICATION SYSTEM USING WEBRTC (RTConn)**

by

Peter Chidubem Atunalu

Computer Engineering Department

European University of Lefke

Supervisor: Assist. Prof. Dr. Zafer Erenel

In a world of computers and cell phones, the need for effective and rapid communication has never been stronger than it is today. RTConn is a real time secured communication system that connects multiple users across the globe using native web API (WebRTC). RTConn allow users connect with each other through video, text, files and screensharing in real time. Allowing collaboration and connection with friends, family, coworker and any one of interest.

By eliminating the need of a server during communication and the requirement to download an external plugin, RTConn increases efficiency, speed and security of data transmitted from one user to another by connecting users’ device together in a pair to pair network.

**Keywords:** WebRTC, VOIP, Internet programming, socket programming, Node.Js, MongoDB, Redis, Video Chat, NoSQL

# Acknowledgments

[OPTIONAL] If you would like to acknowledge any kind of help, moral, advice or motivation from someone. You can also dedicate your work to friends, family ...

# Table of Contents

APPROVAL ii

ABSTRACT iii

ACKNOWLEDMENTS iv

TABLE OF CONTENTS v

LIST OF FIGURES vi

LIST OF TABLES vii

1. INTRODUCTION 1

1.1. DEFINITIONS 2

1.2. GOALS 5

1.3. ETHICS 6

1.4. REQUIRED SOFTWARE 7

2. LITERATURE REVIEW 9

3. BACKGROUND INFORMATION 15

3.1. TITLE OF THE SUBSECTION 19

3.2. TITLE OF THE SUBSECTION 23

4. IMPLEMENTATION DETAILS 24

3.1. TITLE OF THE SUBSECTION 26

3.2. TITLE OF THE SUBSECTION 27

5. CONCLUSION AND FUTURE RECOMMENDATION 29

5.1. BENEFITS 29

5.2. CONCLUSION 32

5.3. FUTURE RECOMMENDATIONS 33

6. REFERENCES 35

APPENDIX A 38

A.1. TITLE OF THE SECTION 38

A.1. TITLE OF THE SECTION 44

APPENDIX B 46

B.1. TITLE OF THE SECTION 49

# List of Figures

Figure 1.1. A Distributed System 8

Figure 1.2. ER Modeling 11

Figure 2.1. xxxx xxxxx 13

Figure 2.2. xxxxxxx 22

Figure 3.1. xxxxx 25

# List of Tables

Table 2.1. Table of parameters 15

Table 3.1. Simulation results depends on parameters 18

Table 3.2. xxxx xxxxx 19

Table 3.3. xxxxxxx 23

Table 4.1. xxxxx 27

# INTRODUCTION

## Definition

### Classic VoIP

Voice over Internet Protocol (VoIP) is a form of communication that allows you to make phone calls over a broadband internet connection instead of typical analog telephone lines. Basic VoIP access usually allows you to call others who are also receiving calls over the internet. Interconnected VoIP services also allow you to make and receive calls to and from traditional landline numbers, usually for a service fee. Some VoIP services require a computer or a dedicated VoIP phone, while others allow you to use your landline phone to place VoIP calls through a special adapter. VoIP is becoming an attractive communications option for consumers. Given the trend towards lower fees for basic broadband service and the brisk adoption of even faster internet offerings, VoIP usage should only gain popularity with time. However, as VoIP usage increases, so will the potential threats to the typical user. While VoIP vulnerabilities are typically similar to the ones users face on the internet, new threats, scams, and attacks unique to IP telephony are now emerging.

Voice over Internet Protocol (VoIP) is a way of voice communication using the Internet Protocol (IP) for the transport. Public Switched Telephone Networks (PSTN1 )is the protocol used by traditional phone networks, and it uses circuit-switching. Information are reserved in the whole communication channel for the duration of the call in Circuit-Switching,. But in packet-switching which is used by the Internet Protocol (IP), data is divided into one or more packets and transmitted digitally. In packet switching every packet knows its origin and destination, and it may travel to it’s destionation via different paths over the network. Developing VoIP requires so many protocols, ranging from those needed to do signaling for call initialization and more, to the transportation of real-time voice across the network, and down to quality-of-service-aware routing, and much more.



VoIP protocols usually uses Real-time Transport Protocol (RTP) as the protocol for media and speech stream. Real-time Transport Protocol (RTP) uses User Data-gram Protocol as the transport protocol. Voice signaling data are usually built using Transmission Control Protocol (TCP) as the transport protocol. Routing and network-level addressing is been provided by the IP layer, while the data-link layer protocols direct and govern the transmission of the data over the physical layer. When choosing between VoIP, VoATM transport, and Layer 2 VoFR integration with other voice or multimedia applications is the major factor of making a decision. When it comes to voice-over-packet in the industry today, and to implementing voice applications VoIP is the predominant form. The complete intranet IP IP infrastructure and internet is used for routing, thus the design any type of calling in a VoIP network becomes simpler. The major problem with VoIP is that it is expensive as compared to Web Real-Time Communication (WebRTC). In this project we will discover the WebRTC API and how it can be larveage to build a real-time communication system.

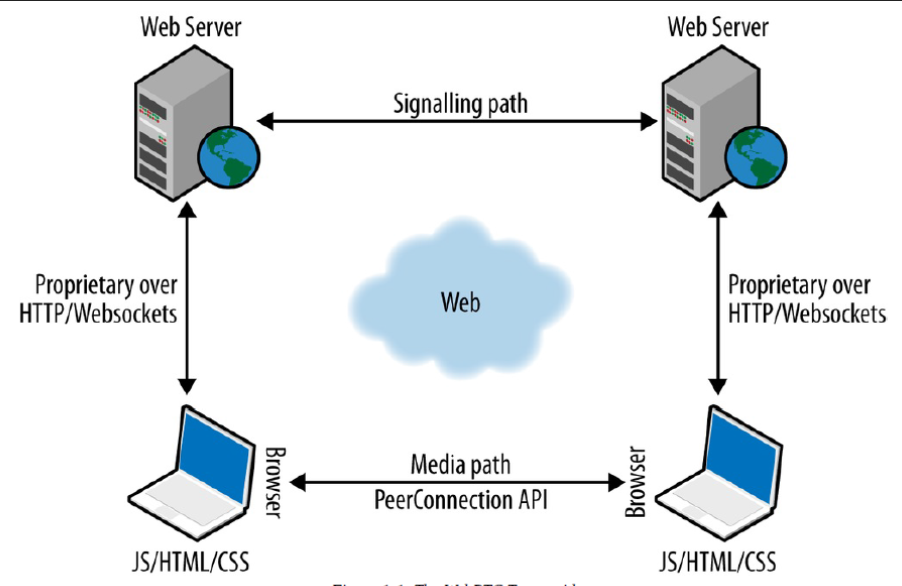
The appearance of VoIP comes at a juncture when telecommunications system has already turned into a large-scale, complex system with multiple, competing infrastructures. VoIP, however, greatly augments the nested complexity by affording a technology that enables multiple architectures and business models for delivering the same voice (and often converged voice and data) service, while remaining agnostic to the underlying infrastructure. The VoIP-enabled architectures have very different capabilities and costs from one another. Many do not – or cannot – support social regulations such as emergency 911, wiretapping and disability access. Most exploit the economic arbitrage opportunities by evading access charges and universal service contributions. Added to this is the combination of reduced asset specificity due to VoIP’s layered architecture and a global standard based ubiquitous IP technology that frees the service providers of the need to own the delivery infrastructure, and enables them to offer service from anywhere globally. Such a misalignment – between regulatory obligations and technical capabilities – has the potential to incubate large-scale systemic failures due to lack of coordination between the local optimization focused private markets and the highly compartmentalized public institutions.

### WebRTC

Web Real-Time Communication (WebRTC) is a new web API which allows browsers to communicate with each other in real time, using a peer-to-peer architecture. WebRTC is a consent-based, audio, video, and data secure peer-to-peer communication API between modern browsers. It enables, organizations, businesses and web developers to develop multimedia real-time applications with no need for external plug-ins; this has never been possible before. WebRTC is an evolution in web applications development, it combines wo historically separated systems together VoIP and web development.

The World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) are jointly defining the JavaScript APIs (Application Programming Interfaces) available at <http://www.w3.org/2011/04/webrtc/> and <https://datatracker.ietf.org/wg/rtcweb/documents/>, the standard HTML5 tags, and the underlying communication protocols for the setup and management of a reliable communication channel between any pair of next-generation web browsers. The standardization aim to define an API that allows a web application that runs on any device, to interchange media and data with a remote party in real-time and in a peer-to-peer fashion through a secure access to the input peripherals of the device (like the webcams and microphones). This will allow for development of several types of application, ranging from a simple audio communication to video-conference with multiple people, providing these functions “out of the box” as part of the basic capabilities of the browser.

In the classic web architecture, an HTTP (Hypertext Transfer Protocol) request for content is sent by the client (browser) to the server, and the server will then respond with a payload that includes all the information the client requested. Uniform Resource Identifier (URI) or Uniform Resource Locator (URL) entity is closely related to the resources provided by the server. The server may include some JavaScript code/file in the web application scenario , when an HTML page is sent to the client. The JavaScript code can interact with the browsers via a standard JavaScript APIs. WebRTC extends the client-server semantics by introducing a peer-to-peer communication paradigm between browsers. The most general WebRTC architectural model draws its inspiration from SIP (Session Initiation Protocol) Trapezoid.



**Fig 1.1:** WebRTC Trapezoid

In the WebRTC Trapezoid model, the web application is been run by the browsers, that is been downloaded from different or same web server. Signaling messages are transported using HTTP or WebSocket protocol via web servers. As needed the server is used to initialize, modify, translate, or manage, and close communications.

WebRTC does not standardized the signaling method to be used between browser and server, leaving developers to implement signaling as part of the application as they see fit. Multimedia flows directly from and to browsers without any intervening servers via the **RTCPeerConnection** API. Direct peer-to-peer connections provides lower latency, thereby making video streaming, text messaging, sensor data feeds, and so on, appear faster. Using a standard signaling protocol such as SIP, XAMP or Jingle (XEP-0166), the two web servers can communicate with each other. Otherwise, they can use a proprietary signaling protocol.

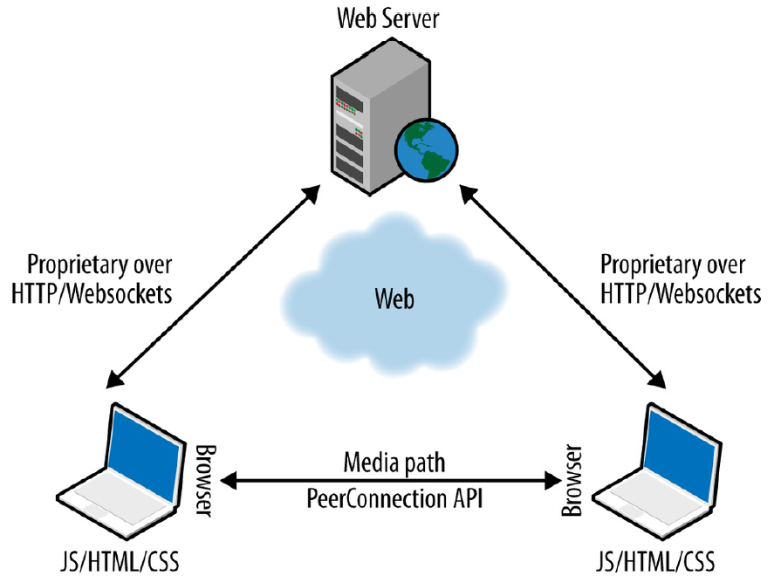


Fig 1.2: The WebRTC Triangle

A WebRTC web application (written as a mix of HTML and JavaScript) interacts with web browsers through the standardized WebRTC API, which allows it to properly discover and control the real-time functions of the browser. RTConn interacts with the browser, using both WebRTC and other standardized APIs, both proactively (e.g., to check browser capabilities) and reactively. WebRTC API provides a wide set of functions, like connection management (in a peer-to-peer architecture), media encoding/decoding, session negotiations, media control, firewall and Network Address Translator (NAT), etc.

**Classic VoIP or WebRTC?**

|  |  |  |
| --- | --- | --- |
|  | VoIP | WebRTC |
| **Signaling** | SIP or H.323 in most cases | Undefined |
| **Media Transport** | RTP/RTCP | RTP/RTCP |
| **Security** | SRTP in SIP,H.235 in H.323 | SRTP |
| **NAT traversal** | STUN/TURN/ICE in SIP,H.450.x in H.323 | STUN/TURN/ICE |
| **Video codecs** | H.263, H.264 | VP8 |
| **Voice codecs** | G.7xx series of codecs, and some more | G.711, iLBC, iSAC |

## GOAL

The aim of this project is to build a real-time communication system that can be deployed on cheap consumer device, while enhancing latency, speed, and security. Now more than ever been able to connect with friends, family, coworkers, instructors, students, and business partners in real-time has become more important than ever before. We were all once together, celebrating life and love, then distance came in and set us apart from our family, friend and work. With so many social media applications like Facebook, Instagram and Whatsapp most of the problem that arise from connect with those we love have been solved. But in other to use this services one most have to sign up for service and be connected with the person you wish to speak with. In other to deliver a real time system that can transmit video companies usually need spend a huge sum of money on hardware, and for this reason most organization depends on other large organization for real time communication.

Some time ago, expensive and complex equipment and expertise is required to develop a video calling or conferencing application. As of today the whole infrastructure required to connect with some remotely anywhere from in the world can be carried in the pocket. With standard Internet connectivity and basic hardware you can easily participate, and host very nice video conferencing sessions using your smart phone and mobile device including your computer. Due to the advent and development of Voice over IP (VoIP), Video conferencing become more common and more accessible to everyone. Video data packets along with voice packets and other types of data (Packet), are transmitted via the Internet, therefore it made voice and video communication free.

The huge benefits of connecting through video calling is a better and efficient meetings with a stronger sense of connection among contacts and the exchange of non-verbal expression and communications, both within and between friends and family, customers, as well as with companies. Face-to-face connection adds non-verbal communication to meetings on a more personal level, the exchange permits participants to build a stronger sense of familiarity with each other, even though they may never be able to meet in person.

Previously, when it came to live video broadcasting, only very well-funded companies where able afford the cost to produce and distribute a working system. This companies usually need need a large audience to make ends needs, due to high production and distribution costs. Therefore they have to broadcast contents for the lowest common denominator; for this reason, regular-season like Little League Baseball are not usually broadcast live — yet. When live video broadcasting was introduced to mobile devices, it revolved the way we develop real-time solution. Bringing down the costs of real-time system to the expense of having a smart phone and an Internet connection, changes people broadcast and what they choose to watch? If we fast-forward to ten years, there will be huge increase in number of services that offers live video broadcasting services; this days there are people on the internet that are sharing everything from cute puppies to books, and other things. A lot of viewers spent more than a billion minutes in total watching “Twitch plays Pokémon” stream.

We are witnessing an increasing number of live broadcasts, breaking news and angles that were never previously available to us. Today even war are streamed live for the world to view. Individuals who are able to creating their own news and sharing it globally are now competing with traditional media companies, with the world as their audience. Broadcasters and publishers have grown interest in mining live video streaming platforms for content they could never have captured themselves, since the advent of consumer live streaming. They mine content created by people like you and me, via modern consumer tech and shared to the world in real-time as an interactive video.

## Required software

WebRTC**:** WebRTC is a free, open project that provides browsers and mobile applications with Real-Time Communications (RTC) capabilities via simple APIs. The WebRTC components have been optimized to best serve this purpose.

**USE:** WebRTC is the tech and backbone of the communication been built in this project.

Node.js:Node.js is an open source,server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Ryan Dahl developed Node.js in 2009. Node.js is a run-time server-side and networking applications used for development of cross-platform application. Node.js applications can be run on any operating that has a Node.js runtime installed and are written in JavaScript.

**USE:** Node.js is used as the server and signaling server for this application. It contains the business logic used in connecting to separate used at remote place apart from each other.

MongoDB: MongoDB is a free and open-source no realtion database (NoSQL) and distributed database at its core, with high availability, horizontal scaling, and geographic distribution are built into it.

**USE:** Used to store user and system data, like username and password.

Socket.io:It is a websocket library written in JavaScript for active bidirectional networking or communication between the server-side (Node.js) and the browser. It abstracts away websockets communication schemes, based upon the capabilities of the browser. With also some convenient features like broadcasts and multicasts, that are beyond the features of the plain html5 websockets.

**USE:** Socket.io is used in this project to transmit signaling message across to all users of RTConnm, and to logically separate user in into rooms and namespace.

Amazon EC2:Amazon Elastic Compute Cloud (Amazon EC2) is a cloud computing service from Amazon. It provides on-demand and salable computing capacity in the Amazon Web Services (AWS) cloud ecosystem. Amazon EC2 cancels out the need for upfront investment in hardware, so development and deployment of applications can become faster and affordable. Amazon EC2 can be used to launch as many or as few virtual servers as need, configure security and networking, and manage compute storage. Amazon EC2 allows applications to scale up or down in order to handle changes in requirements or spikes in popularity, eliminating the need to forecast traffic.

**USE:** EC2 provides the necessary infrastructure need for the back-end and signaling server.

Docker:Docker is a container tool, that makes it easier to design, create, deploy, and run applications. Using containers tool like docker, developers can package up application with all of the parts needed, such as system libraries and application dependencies, and ship it all out as one package. Thanks to the container, we can be rest assured that the application will run on any other Linux machine without any regard to any customization that machine might have and is different from the machine used for writing and testing the application. The statement “it works on my machine” has been eliminated by docker once and for all.

**USE:** Docker help package the application for continuous delivery and deployment pipeline, and makes it easy to deploy the application.

# 2. LITERATURE SURVEY

If you look at the browser, with a few exceptions it currently replaces all of the other platforms and environments for the applications we use. One such exception are applications that require bidirectional voice and video calling capabilities. While you can get these by way of a Flash plugin, this has its drawbacks. WebRTC is a free, open project, supported by Google, Mozilla, Opera and others.

In the beginning of 2010, Google finalized its acquisition of On2, a video codec company that has developed the VP series of codecs, with the latest one being VP8. On2 has always positioned its codecs as a patent free replacement to the H.26x series of codecs, which were standardized, patented and widely used. It then went about opening On2’s technologies to the world and open sources VP8 under the name of WebM. The idea was to replace H.264 for web videos and by that, reduce patent costs for everyone – especially Google itself.

Google went on and during 2010 acquired Global IP Solutions (GIPS), a company known for their media frameworks – a piece of technology that makes developing VoIP and video calling applications easier. At the time, GIPS had a large market share in VoIP, which caused most of the industry to scurry and search for alternative solutions. As with On2, Google took GIPS’ assets and open sourced them. This time, with an interesting twist: they threw out all voice and video codecs that had patent owners and added an additional layer – a JavaScript API as an integration layer to web browsers. The idea? Have bidirectional media processing and media coding technologies available in every browser. It then went on to push it as a standard at the IETF and W3C, where such standards live. It is now called WebRTC.

In the beginning of 2010, Google finalized its acquisition of On2, a video codec company that has developed the VP series of codecs, with the latest one being VP8. On2 has always positioned its codecs as a patent free replacement to the H.26x series of codecs, which were standardized, patented and widely used. It then went about opening On2′s technologies to the world and open sources VP8 under the name of WebM. The idea was to replace H.264 for web videos and by that, reduce patent costs for everyone – especially Google itself. Google went on and during 2010 acquired Global IP Solutions (GIPS), a company known for their media frameworks – a piece of technology that makes developing VoIP and video calling applications easier. At the time, GIPS had a large market share in VoIP, which caused most of the industry to scurry and search for alternative solutions. As with On2, Google took GIPS’ assets and open sourced them. This time, with an interesting twist: they threw out all voice and video codecs that had patent owners and added an additional layer – a JavaScript API as an integration layer to web browsers. The idea? Have bidirectional media processing and media coding technologies available in every browser. It then went on to push it as a standard at the IETF and W3C, where such standards live. It is now called WebRTC."

# 3. BACKGROUND INFORMATION

## Overview

Exchange of arbitrary binary data can be transferred between peers using the **RTCDataChannel** interface. Data channels can be used for back-channel information, metadata exchange, game status packets, file transfers, or even as a primary channel for data transfer. Using JavaScript‐based APIs that enable hardware control and communication through the browser, developers can built a WebRTC application.

## The General Flow

There are a wide range of scenarios, ranging from single web page demos running on a single device to complex distributed multi-party conferencing with a combination of media relays and archiving services. To get started, we will focus on the most common flow, which covers two web browsers using WebRTC to set up a simple video call between them.

Following is the summary of this flow:

• Connect peers

• Initiate signals

• Find candidates

• Negotiate media sessions

• Start RTCPeerConnection streams

### Connect Peers

To connect two users together as a peer, the simplest way of doing this is to have both the users visit the same website. This page can then identify each browser and connect both of them to a shared signaling server, using something like the WebSocket API. This type of web page, often, assigns a unique token that can be used to link the communication between these two browsers.

You can think of this token as a room or conversation ID. In this project, the first user visits https://rtconn.tk, and is then create a unique room (if he create the room eul\_comp the URL for the room will be https://rtconn.tk/eul\_comp). This first user then sends this unique URL to the second user, and once they both have this page open at the same time the first step is complete.

### Initiate signals

Now that both users are in same page, they can now exchange signaling messages to negotiate the setup of their peer connection. In this context, "signaling messages" are simply any form of communication that helps these two browsers establish and control their WebRTC communication as show in Fig 3.1. The WebRTC standards don't define exactly how this has to be completed. This is a benefit, because it leaves this part of the process open for innovation and evolution. It is also a challenge as this uncertainty often confuses developers who are new to RTC communication in general. RTConn application implements this using socket.io. This could, just as easily, be any other approach such as XHR polling, Server-Sent Events (<http://www.html5rocks.com/en/tutorials/eventsource/basics/>),or any combination of these, you feel comfortable with.

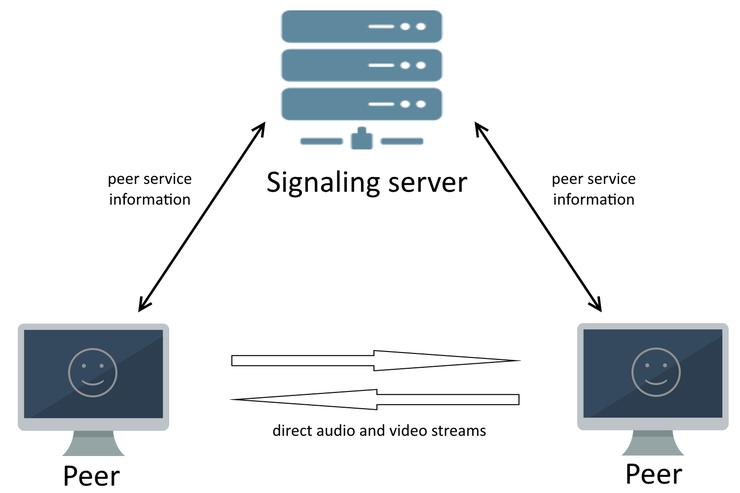


Fig 3.1 RTConn Peer Connection exchanging signaling message

### Find candidates

The next step is for the two browsers to exchange information about their networks, and how they can be contacted, this process is commonly described as "finding candidates". In this stage a directly accessible network interface and port id established at the end each browser. Each browser is likely to be sitting behind a router that may be using Network Address Translation (NAT) to connect the local network to the internet. Their routers may also impose firewall restrictions that block certain ports and incoming connections. NAT Traversal is critical to establishing a WebRTC communication; NAT is finding a way to connect through these types of routers with firewalls that blocks it from accessing the internet [3]. A common and easy way to achieve this is to use a Session Traversal Utilities for NAT (STUN) server [4], which simply helps to identify how you can be contacted from the public internet and then returns this information in a useful form. There are a range of people that provide public STUN servers. RTConn uses one provided by Google and Numb.

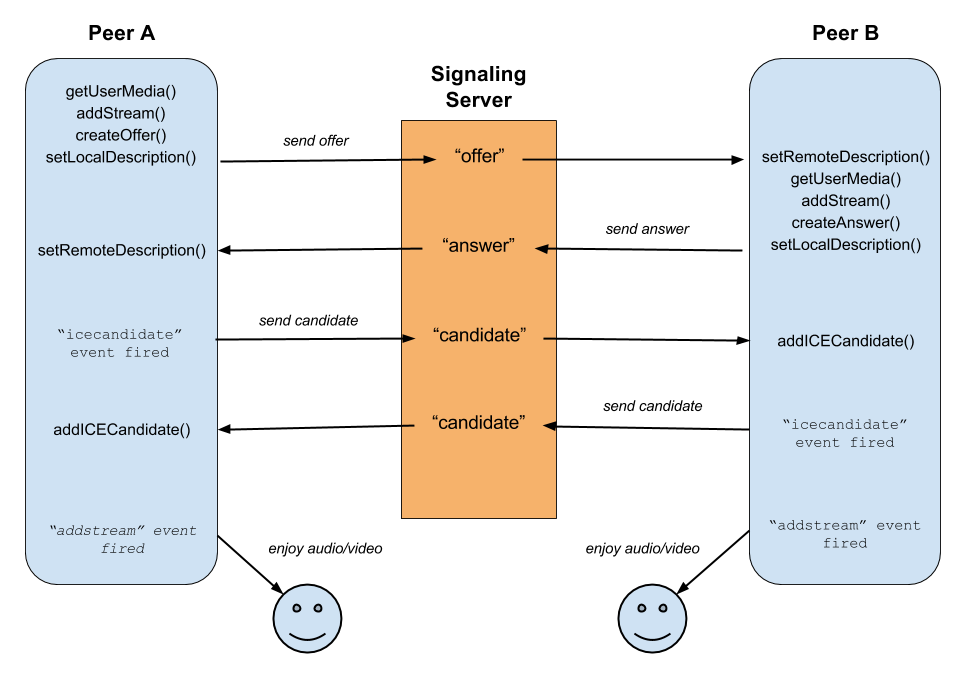
If the STUN server cannot find a way to connect to the browser from the public internet, there is no other option than to fall back to using a solution that relays your media, such as a Traversal Using Relay NAT (TURN) server [5]. This effectively takes you back to a non peer-to-peer architecture, but in some cases, where you are inside a particularly strict private network, this may be your only option. Within WebRTC, this whole process is usually bound into a single Interactive Connectivity Establishment (ICE) framework [6] that handles connecting to a STUN server and then falling back to a TURN server where required.

### Negotiate media sessions

Now that both the browsers know how to talk to each other as they have gain the public information of each other, they now must decide on the type, format codec, resolution, bitrate, etc. of media that will exchanged. This is usually negotiated using an offer/answer based model, built upon the Session Description Protocol (SDP) [7]. This has been defined as the JavaScript Session Establishment Protocol (JSEP) defined by the IETF [8].

### Start RTCPeerConnection streams

Once this has all been completed, the browsers can finally start streaming media to each other, either directly through their peer-to-peer connections or via any media relay gateway they have fallen back to using.

**Fig 3.2: RTCPeerConnection Ready To Stream Live Data**

At this stage show in fig 3.2, the browsers can continue to use the same signaling server solution for sharing communication to control this WebRTC communication. They can also use a specific type of WebRTC data channel to do this directly with each other.

## Signaling

The general idea behind the design of WebRTC has been to fully specify how to control the media plane, while leaving the signaling plane as much as possible to the application layer. The rationale is that different applications may prefer to use different standardized signaling protocols (e.g., SIP or eXtensible Messaging and Presence Protocol [XMPP]) or even something custom. Session description represents the most important information that needs to be exchanged. It specifies the transport (and Interactive Connectivity Establishment [ICE]) information, as well as the media type, format, and all associated media configuration parameters needed to establish the media path. Since the original idea to exchange session description information in the form of Session Description Protocol (SDP) “blobs” presented several shortcomings, some of which turned out to be really hard to address, the IETF is now standardizing the JavaScript Session Establishment Protocol (JSEP). JSEP provides the interface needed by an application to deal with the negotiated local and remote session descriptions (with the negotiation carried out through whatever signaling mechanism might be desired), together with a standardized way of interacting with the ICE state machine. The JSEP approach delegates entirely to the application the responsibility for driving the signaling state machine: the application must call the right APIs at the right times, and convert the session descriptions and related ICE information into the defined messages of its chosen signaling protocol, instead of simply forwarding to the remote side the messages emitted from the browser.

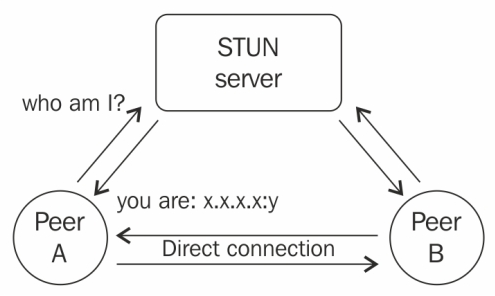
### STUN server

Network Address Translation provides a device with an IP address for use within a private local network, but this address cannot be used externally and without a public address there is no way for WebRTC peers to communicate. To get around this problem WebRTC uses Session Traversal Utilities for NAT (STUN) servers, to try and get an external address to a peer. In a simple world, every WebRTC application would be able to learn its external address which it could exchange to other peers in order to communicate directly. In reality most devices exist behind one or more layers of NAT, firewall, proxies and anti-virus software which may block certain addresses, ports and protocols. STUN is a tool that helps protocols dealing with NAT traversal, it may be used by a device to determine the IP address and port allocated to itself by its NAT. It may also be used to check connectivity between two endpoints, and as a keep-alive protocol to maintain NAT bindings.

A STUN server have one simple task, to check the IP and port of an incoming request from application that is running behind a NAT and send that address back as a response. WebRTC applications can use a STUN server to discover the browsers IP and port from a publi

perspective. This enables a peer to get its own publicly accessible address and then pass it on to another peer via a signaling server in order to set up a direct link.

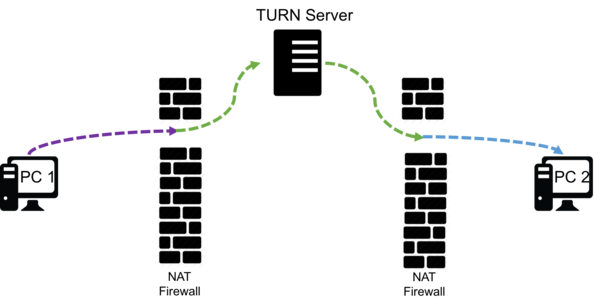
According to Webrtcstats and Bistri statistics that was published in June 2014 1, 92% of all calls successfully made a P2P connection. That means that 8% of the traffic had to be relayed through a TURN server.

**Fig 3.3: RTCPeerConnection Discovering it IP and Port for a STUN server**

### TURN Server

Traversal Using Relays around NAT (TURN) servers is the last resort when trying to create a P2P connection with WebRTC. If a host is located behind proxies, firewalls or strict NAT’s and STUN fails to get the public IP of a peer then it is impossible to set up a P2P connection. When this happens, instead of having the connection fail, WebRTC will fall back to use TURN servers to relay data between the two hosts.

If WebRTC needs to use a TURN server to relay the data, the communication will not be P2P but by using it as fallback WebRTC increases the odds to successfully establish connections for a wide variety of devices, as seen in the illustration shown in Figure 3.4



**Fig: 3.4: WebRTC Connected Using a TURN Server**

TURN servers have a public address so they can be contacted by peers even if the peer is behind a firewall or proxy. They have a theoretically simple task, to relay data between two peers. But unlike STUN servers they will have a huge bandwidth load which results in them having to be sturdier than STUN servers. The TURN servers to be used by the application is specified in the IceConfiguration to the PeerConnection object.

### Using WebSockets

The WebSocket API makes it easy for web developers to utilize bidirectional communication within their web applications. You simply create a new connection using the var connection = new WebSocket(url); constructor, and then create your own functions to handle when messages and errors are received. And sending a message is as simple as using the connection API,  **send(message);**  method.

The key benefit here is that the messaging is truly bidirectional, fast, and lightweight.

This means the WebSocket API server can send messages directly to your browser whenever it wants, and you receive them as soon as they happen. There are no delays or constant network traffic as it is in the XHR polling or long-polling model, which makes this ideal for the sort of offer/answer signaling dance that's required to set up WebRTC communication.

The WebSocket API server can then use the unique room or conversation token, previously described, to work out which of the WebSocket API clients messages should be relayed to. In this manner, a single WebSocket API server can support a very large number of clients. And since the network connection setup happens very rarely, and the messages themselves tend to be small, the server resources required are very modest.

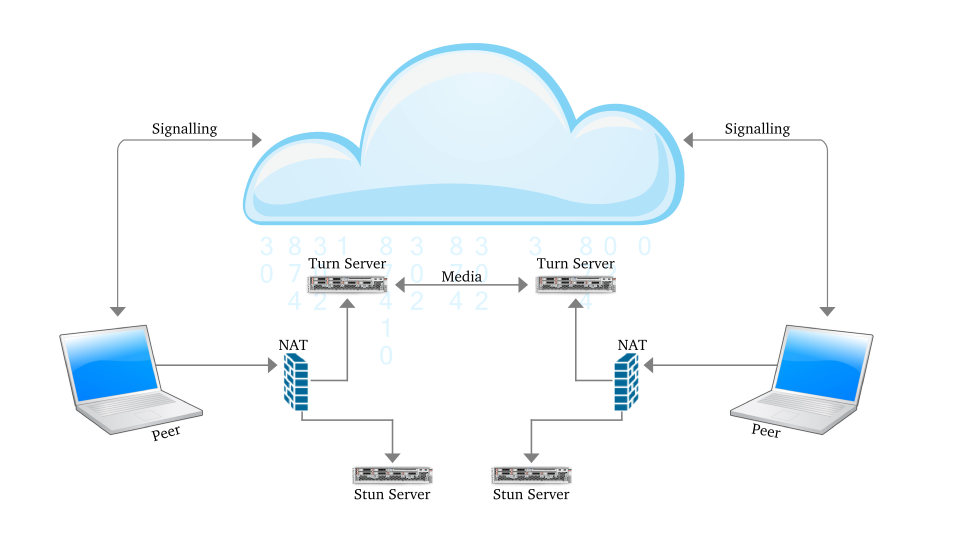
There are WebSocket API libraries available in almost all major programming languages, and since Node.js is based on JavaScript, it has become a popular choice for this type of implementation. Libraries such as socket.io ( http://socket.io/ ) provide a great example of just how easy this approach can really be.

The task of getting the initial signaling data from one peer to another seems like it should be a simple process. Perhaps in a perfect world, a WebRTC signaling mechanism would be able to connect peers directly, without any detours or sidetracking. But the modern internet is structured in such a way that makes this sort of easy relay impossible. NATs of all varieties, and firewalls on many devices, will often erroneously filter packets that are not primed to deal with ALGs and other protective measures. Outside of generating the SDP packet itself, the signaling mechanism is also crucially responsible for ensuring that these signaling messages can be shared between peers in the first place.

So, how does a WebRTC signaling mechanism negotiate the perilous maze of the internet? The answer is simple in theory: it utilizes a versatile framework known as ICE. The efficiency of ICE allows it to calculate, with a mere three methods, the quickest and easiest NAT traversal route for a packet to reach its destination. The first method used, and the least likely to occur, is when ICE tries to make a UDP connection using the host address obtained from a device’s operating system and network card. This will inevitably fail on devices behind NATs, and so there are two remaining methods for ICE to employ: a STUN server or a TURN relay server.

86% of all WebRTC calls are established via STUN servers 1. A STUN server operates STUN servers check the IP address and port of incoming requests, and it then sends that address back to the device’s WebRTC application as a response. The WebRTC application thus uses a STUN server to ascertain its own IP port address from a public perspective. This allows the application to offer a publicly accessible address, which is then passed to another WebRTC-enabled peer via the signaling mechanism.

If both methods fail, the final method employed by ICE is a TURN relay server. TURN servers are used to stream audio, video, and other real-time data between peers. Technically speaking, it does not relay signaling information, because it enables actual real-time data exchanges between peers. TURN servers have publicly available addresses, so peers can connect to them even if they are behind NATs and firewalls. TURN servers are costlier to maintain than STUN servers, because they are actually streaming media rather than connecting peers.



A fully functioning WebRTC application requires all of ICE’s capabilities to operate smoothly and effectively. But purchasing and maintaining numerous servers at a significant cost is simply not a feasible option for developers who are looking to make sound economic and personnel decisions.

## WebRTC APIs

### **GetUserMedia**

The MediaDevices getUserMedia() method prompts the user for permission to use a media input which produces a MediaStream with tracks containing the requested types of media. That stream can include, for example, a video track (produced by either a hardware or virtual video source such as a camera, video recording device, screen sharing service, and so forth), an audio track (similarly, produced by a physical or virtual audio source like a microphone, A/D converter, or the like), and possibly other track types [10].

### **MediaStream**

A MediaStream is an abstract representation of an actual stream of data of audio and/or video. It serves as a handle for managing actions on the media stream, such as displaying the stream’s content, recording it, or sending it to a remote peer. A MediaStream may be extended to represent a stream that either comes from (remote stream) or is sent to (local stream) a remote node. A LocalMediaStream represents a media stream from a local media-capture device (e.g., webcam, microphone, etc.). To create and use a local stream, the web application must request access from the user through the getUserMedia() function. The application specifies the type of media—audio or video—to which it requires access. The devices selector in the browser interface serves as the mechanism for granting or denying access. Once the application is done, it may revoke its own access by calling the stop() function on the LocalMediaStream. Media-plane signaling is carried out of band between the peers; the Secure Real-time Transport Protocol (SRTP) is used to carry the media data together with the RTP Control Protocol (RTCP) information used to monitor transmission management.

### **RTCPeerConnection**

RTCPeerConnection (simply called PeerConnection) is a component that handles multimedia communication between two peers, making sure the streams are stable and efficient. It is an API that contains functions for encryption and bandwidth management. When setting up a connection, both peers need to initialize their PeerConnection object. The configuration parameter is optional and it contains information to find the servers used by Interactive Connectivity Establishment (ICE). There may be multiple servers of each type2 and if no configuration is set, the default settings will be used [2].

### **DataChannel**

RTCDataChannel (simply DataChannel) enables exchange of arbitrary data between two peers with customizable delivery properties of the underlying transport. It is a bi-directional channel and resides as a component of the PeerConnection API [2]. Each application using the channel can configure it to provide the following:

* Reliable or partially reliable delivery of sent messages
* In-order or out-of-order delivery of sent messages

|  |  |  |  |
| --- | --- | --- | --- |
|  | TCP | UDP | SCTP |
| Reliability | Reliable | Unreliable | Configurable |
| Delivery | Ordered | Unordered | Configurable |
| Transmission | Byte-Oriented | Message-Oriented | Message-Oriented |
| Flow Control | Yes | No | Yes |
| Congestion Control | Yes | No | Yes |

**Table 3.1: Comparison of TCP vs UDP vs SCTP**

The standardization work within the IETF has reached a general consensus on the usage of the Stream Control Transmission Protocol (SCTP) encapsulated in DTLS to handle non media data types. The encapsulation of SCTP over DTLS over UDP together with ICE provides a NAT traversal solution, as well as confidentiality, source authentication, and integrity protected transfers. Moreover, this solution allows the data transport to interwork smoothly with the parallel media transports, and both can potentially also share a single transport-layer port number. SCTP has been chosen since it natively supports multiple streams with either reliable or partially reliable delivery modes. It provides the possibility of opening several independent streams within an SCTP association towards a peering SCTP endpoint. Each stream actually represents a unidirectional logical channel providing the notion of in sequence delivery. A message sequence can be sent either ordered or unordered. The message delivery order is preserved only for all ordered messages sent on the same stream. However, the DataChannel API has been designed to be bidirectional, which means that each DataChannel is composed as a bundle of an incoming and an outgoing SCTP stream. The DataChannel setup is carried out (i.e., the SCTP association is created) when the CreateDataChannel() function is called for the first time on an instantiated PeerConnection object. Each subsequent call to the CreateDataChannel() function just creates a new DataChannel within the existing SCTP association

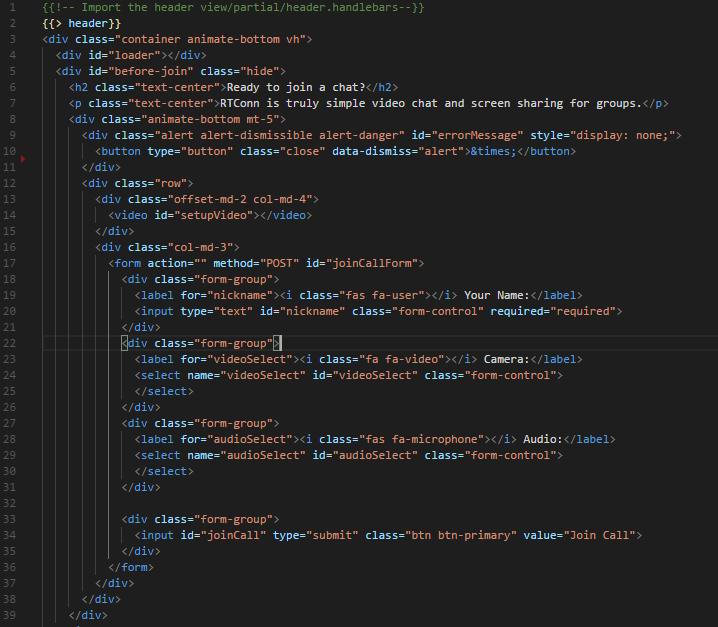
# 4. IMPLEMEMTATION DETAILS

WebRTC allows you to set up peer-to-peer connections to other web browsers quickly and easily. To build such an application from scratch, you would need a wealth of frameworks and libraries dealing with typical issues like data loss, connection dropping, and NAT traversal. With WebRTC, all of this comes built-in into the browser out-of-the-box. This technology doesn't need any plugins or third-party software. It is open-sourced and its source code is freely available at http://www.webrtc.org/. The WebRTC API includes media capture, encoding and decoding audio and video, transportation layer, and session management

## Setup Page

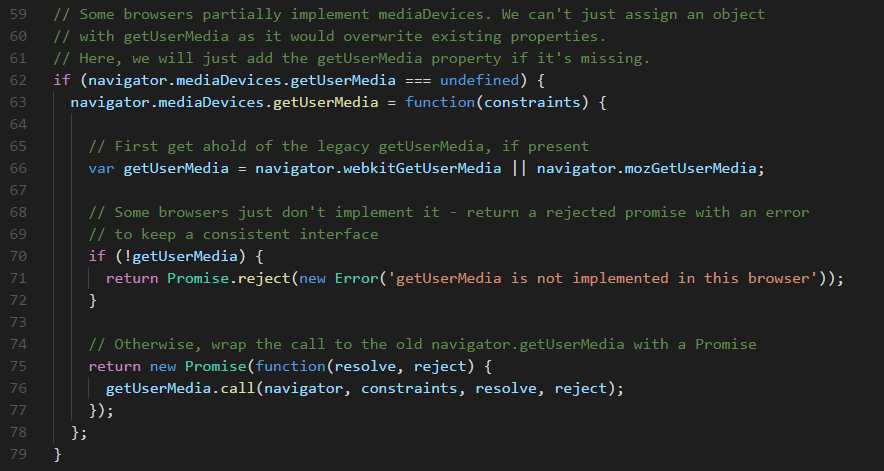
This page get the user ready to join a call session. We will start by capturing the Media Stream and giving the user the opportunity to select their desire camera for video stream and microphone for audio stream, and an input area to input his/her name. Figure 4.1 shows the necessary HTML for the page.

### Step 1: Create HTML



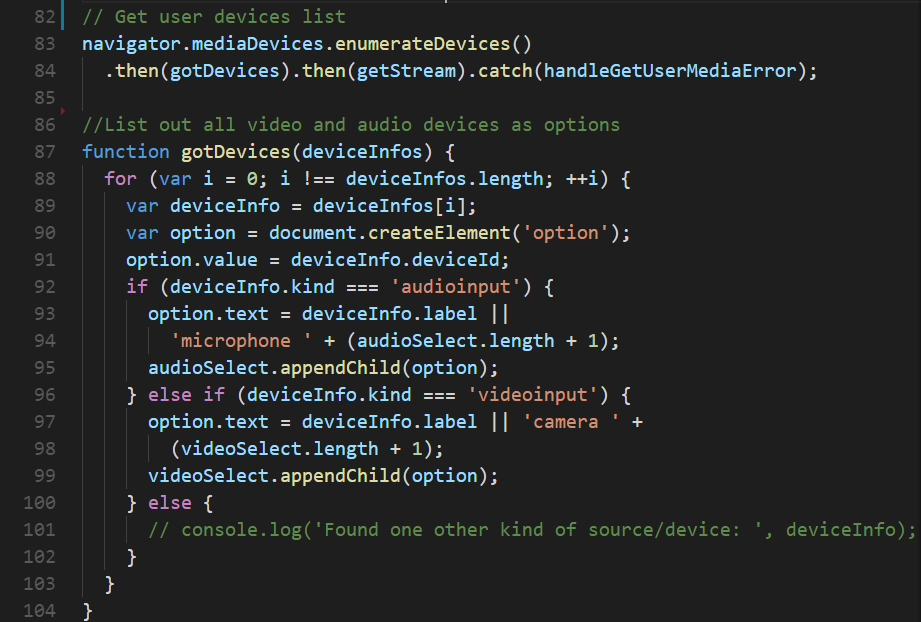
**Fig 4.1: Setup Page HTML**

### Step 2: Prefix getUserMedia API to work on all browser



**Fig 4.2: Make getUserMedia work same on all browser**

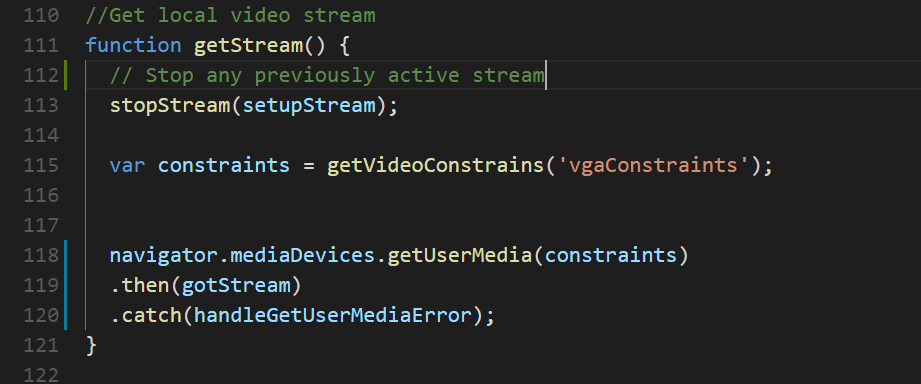
The code above (Fig 4.2) ensure the navigator.mediaDevices.getUserMedia have same API interface regardless of the browser the user is using. This is needed because all browser do not implement this API. First we check to see if it is define, when it is not define the we set it using the legacy getUserMediam if present. We then return a Promise object, on Error or Success.

Step 3: Obtain User Media Devices

**Fig 4.3: Get user media devices and append to select option**

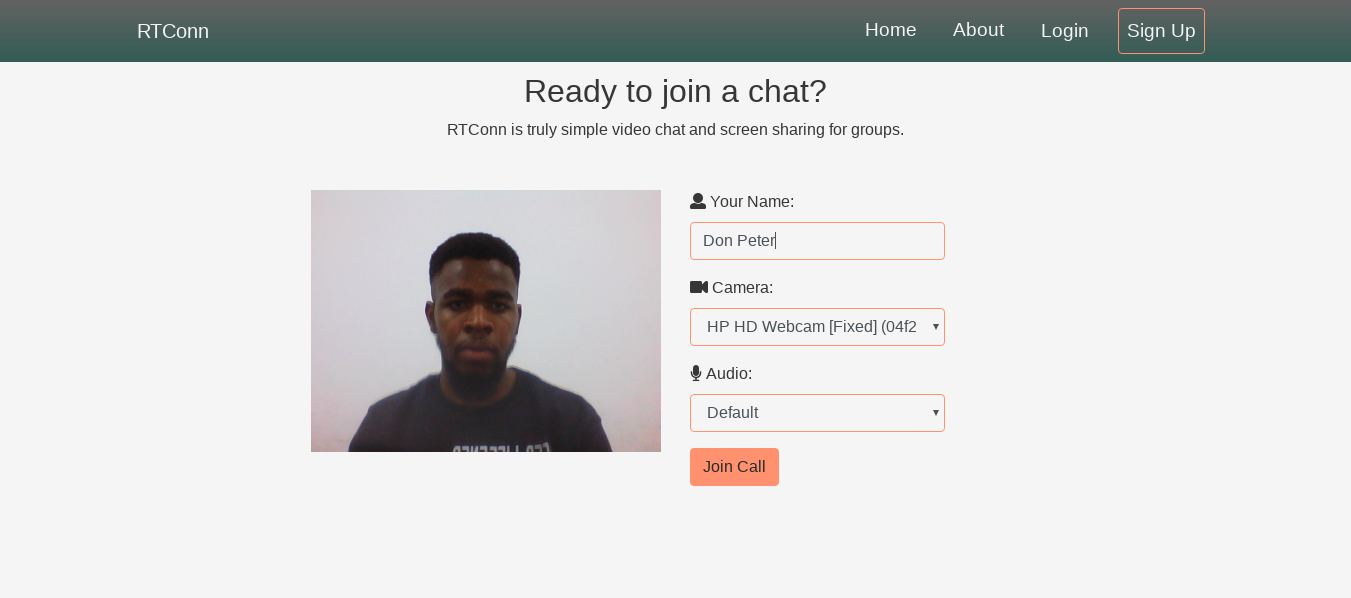
The function above (Fig 4.3) Obtains the list of all media devices, if media type is audio it adds it to the audio option and if it is video it is added to the video option. The value of the option is set to the device ID and is used in the ***getVideoConstrains*** function.

### Step 4: Obtain User Media



**Fig 4.4: Obtain User webcam and audio function**

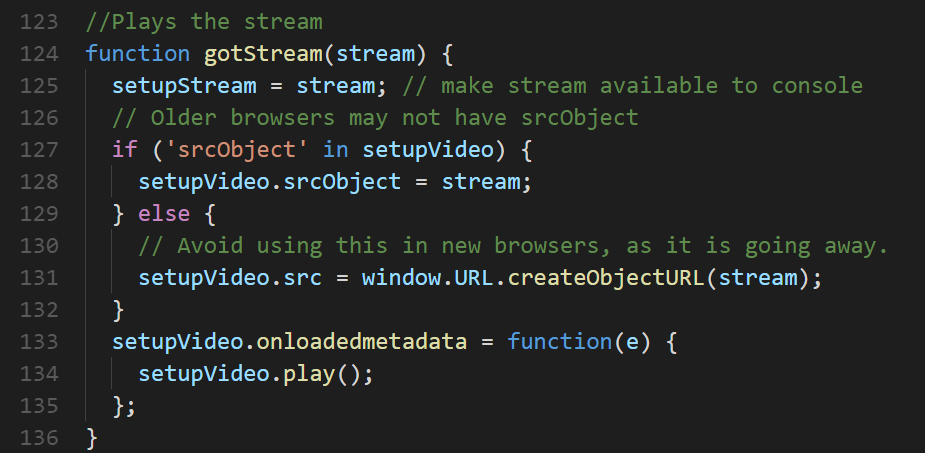
The ***getStream*** function (Fig 4.4) fires every time the select video or audio select box on the setup page changes. First any previously selected stream is stopped, and then a new constrain for the stream is obtained. Once the stream is stop and the new constrain is obtained we the call the ***getUserMedia*** function passing the new constrain, this function may requires user permission Fig 4.5 shows this in a Chrome browser.



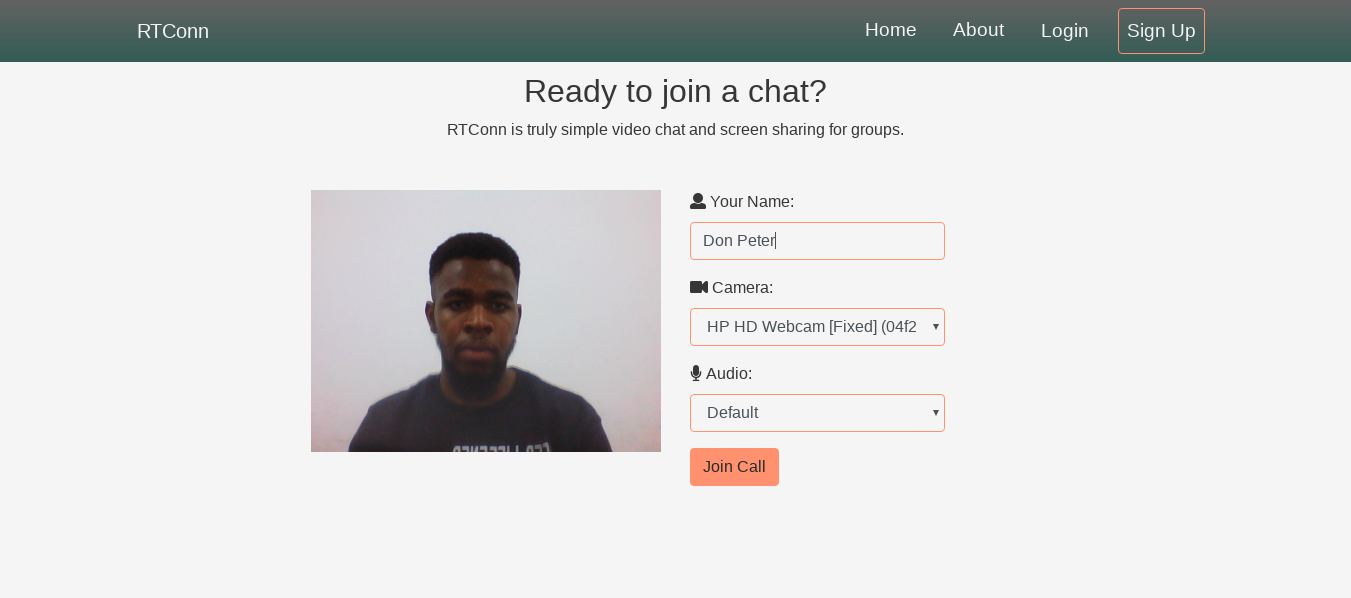
**Fig 4.5: getUserMedia Permission box.**

### Step 4: Attach Stream to Video Element Source

Once we obtain user media stream next we have to attach it to our local video stream. Figure 4.6 show the code for this and figure 4.7 show the page after it has been attached.



**Fig 4.6: Attach the stream to the setup video src**



**Fig 4.7: Setup Page After User Stream has been attach**

chapter also should contain a scheme of the implementation of your method or solution, organization of experiments, based on computation, simulation, and statistical analysis, with appropriate graphs and other illustrations. The obtained results must be discussed and compared with the results given in published works.

Chapter 3 and this chapter together represent the most important parts of your new contribution to the area of study. If necessary, the material of this chapter can be divided into two chapters.

# 5. CONCLUSION AND FUTURE RECOMMENDATION

This Chapter can conclude the project. It should summarize the results of study, emphasize their positive and negative aspects and suggest directions of a further study of the topic to improve the proposed scheme, method or approach.

# 6. REFERENCES

**Example of References**

*Books:*

[1] Real-Time Communication with WebRTC, Salvatore Loreto and Simon Pietro Romano, 2014 pp. 15–25.

[2] Peer-to-peer communication in web browsers using WebRTC, 2015, pp. 15–19.

Webs:

[3] <http://en.wikipedia.org/wiki/NAT_traversal>

[4] <http://en.wikipedia.org/wiki/Session_Traversal_Utilities_for_NAT>

[5] <http://en.wikipedia.org/wiki/Traversal_Using_Relay_NAT>

[6] <http://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment>

[7] <http://en.wikipedia.org/wiki/Session_Description_Protocol>

[8] <http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-00>

[9] <https://www.onsip.com/blog/webrtc-signaling>

[10] <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia>

*Journals(Periodicals):*

[3] J. U. Duncombe, “Infrared navigation—Part I: An assessment of feasibility,” *IEEE Trans. Electron Devices,* vol.ED-11, pp. 34–39, Jan. 1959.

[4] E. P. Wigner, “Theory of traveling-wave optical laser,”*Phys. Rev.,* vol. 134, pp. A635–A646, Dec. 1965.

[5] E. H. Miller, “A note on reflector arrays,” *IEEE Trans.Antennas Propagat.,* to be published.

*Articles from Conference Proceedings (published):*

[6] D. B. Payne and J. R. Stern, “Wavelength-switched passively coupled single-mode optical network,” in *Proc.IOOC-ECOC,* 1985, pp. 585–590.

*Papers Presented at Conferences (unpublished):*

[7] D. Ebehard and E. Voges, “Digital single sideband detection for interferometric sensors,” presented at *the 2nd Int.Conf. Optical Fiber Sensors*, Stuttgart, Germany, Jan. 2-5, 1984.

*Standards/Patents:*

[8] G. Brandli and M. Dick, “Alternating current fed power supply,” U.S. Patent 4 084 217, Nov. 4, 1978.

*Technical Reports:*

[9] E. E. Reber, R. L. Mitchell, and C. J. Carter, “Oxygen absorption in the Earth’s atmosphere,” Aerospace Corp.,Los Angeles, CA, Tech. Rep. TR-0200 (4230-46)-3, Nov. 1968.

**Example of E. References—Electronic Sources**

*Books:* Author. (year, month day). *Title.* (edition) [Type of medium]. *volume (issue).* Available: site/path/file

*Example:*

[1] J. Jones. (1991, May 10). *Networks.* (2nd ed.) [Online].

Available: http://www.atm.com

*Journals:* Author. (year, month). Title. *Journal.* [Type of medium]. *volume (issue),* pages. Available: site/path/file

*Example:*

[2] R. J. Vidmar. (1992, Aug.). On the use of atmospheric plasmas as electromagnetic reflectors. *IEEE Trans.Plasma Sci.* [Online]. *21(3),* pp. 876–880. Available:

http://www.halcyon.com/pub/journals/21ps03-vidmar

*Papers Presented at Conferences:* Author. (year, month). Title. Presented at Conference title. [Type of Medium]. Available: site/path/file

*Example:*

[3] PROCESS Corp., MA. Intranets: Internet technologies deployed behind the firewall for corporate productivity.Presented at INET96 Annu. Meeting. [Online]. Available:

http://home.process.com/Intranets/wp2.htp

*Reports and Handbooks:* Author. (year, month). Title.Company. City, State or Country. [Type of Medium]. Available: site/path/file

*Example:*

[4] S. L. Talleen. (1996, Apr.). The Intranet Architecture:Managing information in the new

paradigm. Amdahl Corp., CA. [Online]. Available:

http://www.amdahl.com/doc/products/bsg/intra/infra/html

# APPENDIX

(if any) can contain additional material, such as source texts of programs, large tables of obtained results, descriptions of used protocols, utility programs, etc. Each Appendix must have its title on a separate page.

1. SPECIFIC REQUIREMENTS

For printing and paper quality, use of laser printers are recommended. Paper size should be A4 size (21 x 29.7 cm), portrait (vertical) orientation. But student can use landscape (horizontal) orientation only for a special purpose. The thesis must be printed on good quality white paper on one side of the paper only.

The text must be typed preferably in Times New Roman 12 pt. font with a margin of 4 cm on the left (the binding side). The remaining margins must be 2.5 cm. wide. Text must be one and a half-spaced, except for quoted sections, references, footnotes, and captions of tables and figures.

The page numbers in the preliminary material are to be in lower case Roman numerals, starting with the approval page that is numbered “ii”. Title page is unnumbered but is the implied number “i”. First page of Chapter 1 (Introduction) uses the Arabic number “1” and pages thereafter carry consecutive Arabic numbers, including the pages in the Appendices and the References. All page numbers are positioned in the upper right-hand corner and 1.5 cm above the first line within the required margin boundaries.

Each chapter must start with a new page. Chapter titles should be written by uppercase letters, with a number ahead.

Titles of section should be written in low case letters, with larger font size (12 point-size) than in the text. The first letter of each word of a section title must be in uppercase. A new section should start immediately after the previous section.

Before the title of a new section (subsection) in the text, there should be 2 blank lines. After the title of a section (subsection), there should be 1 blank line. The same font style should be used for titles of chapters and sections. Do not make the titles of chapters and sections italic and do not underline them and any part of the text. Never use color in the text.

The size of a section (or a subsection) should not be shorter than one page. The same is true for Introduction and Conclusion. Giving a list of some items, do not use bullets since they cannot be easily referenced in the text. If you give a list of some items, then number these items or assign a letter (in brackets) to each listed item. There should be no announcement-like statements or emotional expressions in the text.

Divide each section into paragraphs. All subsequent paragraphs and paragraphs of a section must have 3 – 5 blank spaces at the beginning of the first line. A paragraph should not be longer than 1/3 of a page.

If there is a mathematical expression in a separate line, it must be separated from text above and below by two blank lines. Each mathematical expression, written on a separate line, must be numbered on the right side. The numeration should include the chapter number, for example, (5.7). Here 5 is a chapter number and 7 is the number of expression in this chapter.

An example of equation layout

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx is

|  |  |  |
| --- | --- | --- |
|  |  | (5.7) |

All figures must be centered and properly labeled. They must be drawn by using a proper computer software of your own choice. Each figure must have its numbered title (caption) written under the figure. Each table must have its numbered title written over the table. The figure captions should be printed using a 10-point proportional font and italic such as Times New Roman with single line spacing. Number figures and tables in each chapter separately (for example, Figure 4.1, Figure 5.1, Table 3.2, etc.). Each figure and table must be referenced in the text.

As shown in Figure 1.6, xxxxxxxxxxx.



Figure 1.6 The view

As shown in Table 3.2, xxxxxxxxxxxxxxxxxxx.

Table 3.2. Data and parameters

|  |  |
| --- | --- |
| Xxxx | Xxxx |
| Xxxx | Xxxx |
| Xxxx | Xxxx |
| Xxxx | Xxxx |

The total number of figures and tables in the project should be not less than 10. There is no good project without a sufficient number of figures, graphs, tables etc. Avoid using primitive (not informative) figures.

Do not put a figure or a table at the start or at the end of the section, in which it is referenced. That is, each figure or a table should be surrounded by the text of the corresponding section.

On graphs, show only the coordinate axes, or at most the major grid lines, to avoid a dense result after reduction. Do not put boxes around your figures to enclose them.

In the text, use only short fragments of programs if they are necessary for explanation. These fragments must be given as figures in the text. As a rule, all complete source texts of programs must be placed in Appendices.

Graduation Project should contain not less than 15 references, preferably from journal and conference proceedings. Each reference must be referenced at least once in the text of the project. The list of references should be given in the order of citation in the text, not in alphabetical order. To reference a source with number 6 in the list of references, use [6].

Formatting of the project, such as the format of the title page, font style, interline spacing etc., must be done according to the requirements of the engineering department. If some items are not specified in those requirements, then the student should consult his/her supervisor.