Technology Case Study on Web Real-Time Communications
(WebRTC)

A Master’s Project
Presented to
Department of telecommunications

In Partial Fulfillment
Of the Requirements for the
Master of Science Degree

State University of New York
Polytechnic Institute
By
Nagarjuna Karnati
May 2016
Technology Case Study on Web Real-Time Communications (WebRTC)

Declaration

I declare that this project is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Nagarjuna Karnati

05/11/2016
SUNYPOLY
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

Approved and recommended for acceptance as a project in partial fulfillment of the requirements for the degree of Master of Science in Telecommunications

7/19/16
DATE

Dr. Larry J. Hash
Project Advisor
ABSTRACT

Web real-time communication (WebRTC) is the latest technology standard which enables web browsers to communicate directly without having to install any internal or external plug-ins. WebRTC fills a critical gap in the web platform where a native proprietary app like Skype could do something which is media communication that World Wide Web just couldn’t. Now, this can be done form web using WebRTC technology. This paper starts with a brief introduction of WebRTC and how it got started. Moving on, it provides information about the WebRTC technical goals, architecture and protocols involved. This paper highlights the network address translation (NAT) traversal where STUN, TURN and ICE protocols are involved. Also, this paper highlights about the peer to peer to media flows with reference to WebRTC protocol stack and application program interface (API). In the end, this paper discusses about implemented security features, tools available for WebRTC development and provides enterprise use cases.
# Table of Contents

ABSTRACT ......................................................................................................................... 05

CHAPTER 1: INTRODUCTION ............................................................................................ 10

CHAPTER 2: BACKGROUND ................................................................................................. 11

CHAPTER 3: LITERATURE REVIEW ...................................................................................... 12

CHAPTER 4: STANDARDS & DEVELOPMENT OF WebRTC .................................................. 15

CHAPTER 5: AUDIO&VIDEO ENGINES .................................................................................. 16

5.1 Acquiring audio and video with getUserMedia ............................................................... 17

CHAPTER 6: REAL-TIME NETWORK TRANSPORTS ........................................................... 19

CHAPTER 7: PEER TO PEER CONNECTION ESTABLISHMENT ........................................... 21

7.1 Network Address Translator .......................................................................................... 21

7.2 Signaling & session negotiation ...................................................................................... 23

7.3 Selecting a Signaling Service .......................................................................................... 24

7.4 Google Channel API ...................................................................................................... 25

7.5 Session Description Protocol (SDP) ................................................................................ 27

7.6 Interactive Connectivity Establishment (ICE) ............................................................... 28

7.7 STUN binding request .................................................................................................... 31

7.8 Options for Server Setup/Hosting .................................................................................. 34

CHAPTER 8: DELIVERING MEDIA AND APPLICATION DATA ............................................ 35

8.1 Secure Communication with DTLS .................................................................................. 35

8.2 Delivering media with SRTP&SRTCP .......................................................................... 37

8.3 Delivering application data with SCTP .......................................................................... 40

CHAPTER 9: DATACHANNEL ............................................................................................... 44
CHAPTER 10: MULTIPARTY ARCHITECTURE .......................................................... 45
CHAPTER 11: SECURITY IN WebRTC ........................................................................ 47
CHAPTER 12: USE CASES AND INSPIRATION ................................................................. 48
CHAPTER 13: CONCLUSIONS ...................................................................................... 50
  13.1 RECOMMENDATIONS ..................................................................................... 50
REFERENCES ............................................................................................................. 51
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Full WebRTC Environment</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>WebRTC voice and video engines</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>MediaStream carrying synchronized tracks</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>WebRTC protocol stack</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>Reserved Private IP ranges</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>IP Network Address Translator</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>Peers behind NAT device</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>WebRTC Shared signaling channel</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>Channel Message Request</td>
<td>26</td>
</tr>
<tr>
<td>10</td>
<td>Channel Message Response</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>Sample SDP offer</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>STUN request for public IP and port</td>
<td>29</td>
</tr>
<tr>
<td>13</td>
<td>TURN relay server</td>
<td>29</td>
</tr>
<tr>
<td>14</td>
<td>ICE Checklist</td>
<td>30</td>
</tr>
<tr>
<td>15</td>
<td>Peer to peer media using STUN configuration</td>
<td>31</td>
</tr>
<tr>
<td>16</td>
<td>Media flow through TURN</td>
<td>31</td>
</tr>
<tr>
<td>17</td>
<td>Format of STUN Message Header</td>
<td>32</td>
</tr>
<tr>
<td>18</td>
<td>STUN configuration</td>
<td>33</td>
</tr>
<tr>
<td>19</td>
<td>Sample DTLS handshake</td>
<td>36</td>
</tr>
<tr>
<td>20</td>
<td>SRTP header</td>
<td>38</td>
</tr>
<tr>
<td>21</td>
<td>Comparing TCP vs. UDP vs. SCTP</td>
<td>41</td>
</tr>
<tr>
<td>22</td>
<td>SCTP header and data chunk</td>
<td>41</td>
</tr>
</tbody>
</table>
Figure 23. Sample SDP snippet for SCTP association ...................................................... 44
Figure 24. Distribution architecture for an N-way call ...................................................... 45
Figure 25. WebRTC Secure Pathways .............................................................................. 47
CHAPTER 1: INTRODUCTION

WebRTC is a combination of standard protocols and API's which enables browser to browser voice, video communication and arbitrary data transfer. This technology brings real-time communication capabilities for any web application without having to install any third party software or plug-in. A lot of new functionality needs to be added for bringing real time communication capabilities to a browser. All of this new functionality and complexity will be hidden in three primary API's for making it easy for web developers while building applications.

Below are the three primary API's defined by W3C.

MediaStream API: The main function of this API is to capture audio and video streams.
RTCPeerConnection API: Function of this API is to communicate media (audio, video) data.
RTCDATAChannel API: Function of this API is to communicate arbitrary application data.

The WebRTC API appears to be simple, which is handed over to developers, but there will be a lot of components involved together such as signaling, discovering peers, negotiating connection, security, new layers of protocols and so on. WebRTC architecture combined with protocols defines the performance characteristics like connection initiation setup, network routing information and packet delivery. Unlike regular web browsing model WebRTC uses user datagram protocol (UDP) for data transport. However, UDP is the initial stage protocol that gets involved during real-time communication, but there will be a lot more protocols defined on top of UDP to make this technology work better. WebRTC support has already been running on 1Billion user’s devices as it is provided in the latest Chrome and Firefox browsers.
CHAPTER 2: BACKGROUND

Before WEBRTC, macromedias Flash communication server MX (2002) has been used for communication over IP network. This later acquired by adobe and changed the name from FCS to Flash Media Server which is now being called as Adobe Media Server. FCS allowed developers to capture webcam video and audio, stream it right from browser to the server and then to multiple viewers anywhere in the world. FCS has provided tunneling mechanism to traverse firewalls which worked really well on 98% of PC's.

Eventually an open source alternative red5 released and commercial competitor WOWZA released, a server that matches the functionality of adobe.

FCS has everything such as flashes API to incorporate audio and video features into browser except for a workable licensing model. The licensing cost of a single FCS server will cost around $4000 and it provides a limited number of connections per server. Moreover, the challenges were to traverse firewalls, initiating a handshake and lower latency for video and audio over the web.

WebRTC is an open source and it is free. It all started in the year 2010 by Google, Apple, Microsoft, Mozilla, Ericsson, for building a real time communication platform that would work across the browsers without plug-ins.

IETF (Internet Engineering Task Force Organization) RTC WEB Working Group and W3C (World Wide Web Consortium) WebRTC Working Group were formed to start defining the Specifications.
CHAPTER 3: LITERATURE REVIEW

References used in this document are IETF-RTCWEB papers (Internet Engineering Task Force Organization), W3C-WebRTC working draft (World Wide Web Consortium) and some published books.

Major resources referred throughout this paper are IETF-RTCWEB standards draft-ietf-rtcweb-overview-15, RFC 5766, draft-ietf-rtcweb-rtp-usage-25, draft-ietf-rtcweb-data-channel-13, and draft-ietf-rtcweb-security-08.

H. Alvestrand is the author of the paper draft-ietf-rtcweb-overview-15 published by IETF. He works for Google and also a member of the IETF Network Working Group. Author has presented an overview of real time communication on the web by explaining about its architecture and protocols involved. In this paper author has clearly explained the relationship between WebRTC API and the network protocol suite behind this for bringing real time communication capabilities to web browsers. Also, author has summarized various things involved in implementing real-time communication, such as data framing, securing data, transporting data, managing connections and security considerations. This paper provides an overview of WebRTC to get an understanding of things involved and also, author has referenced papers for further reading on every major topic.

R. Mahy, P. Matthews and J. Rosenberg are the authors of paper IETF RFC5766 which explains about traversing firewalls and other NAT devices for implementing direct peer to peer connection. For this, the authors explained an approach called Interactive Connectivity Establishment (ICE), which makes use of STUN and TURN protocols as part of ICE. In this paper author discusses about how STUN and TURN servers can be configured in browsers for implementing direct peer to peer communication which provides in depth understanding of STUN and TURN protocols and ICE approach.

C. Perkins, M. Westerlund and J. Ott are the authors of the paper draft-ietf-rtcweb-rtp-usage-25 which provides a great detail of Media transport and use of real time transport protocol in WebRTC. C. perkins works at the University of Glasgow, M. Westerlund Works for Ericsson and J. Ott works at Aalto
University. The authors of this paper are also the members of RTCWEB working group. In this paper, the authors have provided great detail on how WebRTC makes use of real time transport protocol components such as payload formats, RTP sessions and header extensions for improving data transport robustness. This paper discusses only about the secure profile of RTP because encryption has become mandatory for WebRTC implementation and SRTP is the protocol used for transporting media such as audio and video. In a nutshell, this paper provides great detail of how media gets transmitted behind RTCpeerconnection API, once the connection is established.

R. Jesup, S. Loreto and M. Tuexen are the authors of the paper draft-ietf rtcweb-data-channel-13 which explains about the protocols involved behind WebRTC data channel API. R. Jesup works for Mozilla, S. Loreto works for Ericson and M. Tuexen works at Muenster University of Applied Sciences. The authors of this paper are also the members of the IETF Network Working Group. In this paper authors have provided a great explanation of how non-media data gets transported in WebRTC framework. This paper discusses about WebRTC data channel requirements and then how Stream Control Transmission Protocol (SCTP) can be used for exchanging generic data from peer to peer.

E. Rescorla is the author of the paper draft-ietf rtcweb-security-08 which provides great detail of WebRTC security considerations. In this paper, author defines a security threat model and survey security threats for the implemented model. The interesting thing about this paper is that the author has discussed security in each WebRTC component from its architectural point of view such as while accessing user local devices, threats from screen sharing, while making calls and performing user verification for communication.

Another major resource referred for writing this paper is a published book “High Performance Browser Networking” written by Ilya Grigorik. He works as web performance engineer at Google. In this book author discusses about the network of things behind browser, starting from fundamental limitations to powerful innovations across browser applications such as HTTP 2.0, WebSocket and Peer to peer communication with WebRTC. In this book, author explains about networking protocols (TCP, TLS, UDP, HTTP and many more) and their performance characteristics for building powerful web applications. This
book also answers a lot of questions about networking protocols such as why TCP isn’t good for transporting media when compared to UDP, why latency is the major problem for better performance and how bandwidth management can be achieved by making reuse of network connections etc. After explaining about browser networking foundations, author has discussed the latest advancements in protocols and browser such as benefits of HTTP 2.0 standard, Use of Websocket for building data channels, and building low latency video conference applications using real-time WebRTC transports.

The next major reference used for writing this technology case study is “WebRTC APIs and RTCWEB Protocols of the HTML5 Real-Time Web”, Edition 3.0. Alan B. Johnston and Daniel C. Burnett are the authors of this book which provides information about the architecture, protocols, application program interfaces (APIs) and technical goals of WebRTC. Dr. Alan B. Johnson works as a distinguished engineer at Avaya, Inc. and he is also working as adjust professor at Washington university in St Louis. Daniel C. Burnett works as a chief scientist at Tropo and he is also performing duties as Director of Standards at Voxeo. Sam Dutton who works as a developer advocate for Google Chrome referred this book as a bible for learning about WebRTC. This book provides great details about various network topologies and signaling pathways involved for WebRTC development.
CHAPTER 4: STANDARDS & DEVELOPMENT OF WebRTC

Brining real-time communication capabilities within the browser is a very important addition to the web platform since its evolution. The network layer in the browser has gone through a full re-engineering process as WebRTC breaks the familiar client to server communication model. Therefore, WebRTC architecture has dozen over different standards which covers application API's, protocols and data formats. IETF RTC WEB Working Group and W3C WebRTC Working Group were formed to start defining the Specifications. WebRTC standards have been designed in such a way that communication is not just limited between browsers but can also be incorporated with existing communication systems such as PSTN, SIP end clients, VoIP and many more as shown in the Figure 1 below. Imagine if computers, smartphones, TV’s, Home appliances, landline phone could all communicate on a common platform. Today WebRTC is enabled in 1 billion end points. Also, the main aim of this standard is to bring web capabilities to the telecom world which will be a huge industry growth. This technology is much more than just an API.

Figure 1: Full WebRTC Environment
CHAPTER 5: AUDIO&VIDEO ENGINES

In order to provide teleconferencing experience in browsers at first the system hardware needs to be accessed to capture audio and video. However, the acquired raw media streams won't be good enough for proper communication. Each media stream needs to be processed for better quality and synchronization. Moreover, the output stream bit rate must get adjusted with fluctuating bandwidth between clients.

Now, the media streams received by other client needs to be decoded properly in real time. In a nutshell, capturing and processing media streams is a complicated task. But, WebRTC brings full featuring audio and video engines into web browser which takes care of a lot of things for successful communication as shown in the Figure 2 below.

![WebRTC voice and video engines](https://webrtc.org/architecture/)

*Figure 2: WebRTC voice and video engines*
*Retrieved from: https://webrtc.org/architecture/*
At first the media stream retrieved from system hardware needs to be processed for echo cancellation and noise reduction for good quality. Next the media stream will be encoded using an optimized wideband codec. Furthermore, a unique error concealment algorithm will be utilized for removing any network performance effects like packet loss and network latency.

All of the above-mentioned steps will be handled directly by browser and it can also adjust its processing pipeline dynamically for continuous fluctuating parameters of media, streams and network conditions. An optimized media stream will be sent to web application only after it has gone through above-mentioned processing techniques.

5.1 Acquiring audio and video with getUserMedia

This is the primary API which empowers the web application to procure media streams from the system platform to manipulate and process through further steps.

A MediaStream object can have zero or more individual tracks which will always be in synch from one another. These tracks are known as MediaStreamTracks. A media stream input source can be voice, video or any arbitrary data from a hard drive. Using getUserMedia API, the processed media streams can be sent to the next set of API's for further processing.

A MediaStream object performs activities such as acquiring data and manipulating individual tracks.

![Figure 3: MediaStream carrying synchronized tracks](http://io13webrtc.appspot.com/#16)
Sometimes, the input source capabilities may restrict MediaStream features such as a few clients, webcams can create higher definition video streams than others. Therefore, while requesting for media streams from a browser this API permits us to indicate compulsory or optional constraints depending on our need. The next step is to route the acquired data to destination peer.
CHAPTER 6: REAL-TIME NETWORK TRANSPORTS

Real-time communication is time-sensitive because it is more important that the information has sent on time to the receiver rather than guaranteeing its delivery. While looking at an existing audio video streaming apps one can observe that these have been designed tolerant to packet loss and output quality. If needed applications has to implement their own logic to overcome packet loss and delay in packet transport. Therefore, low latency and timeliness are significantly more critical than reliability for implementing successful real time communication.

In order to meet the above-mentioned requirements, UDP has been preferred over TCP for data transport in real time communications. TCP provides reliable data transport where if packet loss occurs then TCP doesn't continue sending remaining packets instead it buffers all the packets after the lost packet and waits for retransmission until it delivers them in an orderly manner to the application. By comparing UDP with TCP, it differs from the following services.

- UDP doesn't guarantee message delivery which means no acknowledgment, no retransmission.
- UDP doesn't guarantee packets being delivered orderly which means no packet sequencing, no reordering.
- UDP doesn't track for connection status.
- UDP doesn't provide congestion control which means there's no built-in network feedback mechanism.

The transport layer of the WebRTC uses UDP which delivers the packets the moment they arrive with no sequencing or ordering. This UDP alone in the transport layer won't be enough for implementing successful real time communication. Several other mechanisms along with protocols should be implemented for many other activities like traversing many layers of NAT's and firewalls, negotiating each stream parameter, implementing flow control, providing data encryption and many more.
UDP is the basis for implementing real-time communication in web browser, but will also need a large supporting cast of protocols on top of UDP to meet requirements as shown below in Figure 4.


In order to set up and maintain peer to peer connection, ICE STUN and TURN must be implemented on top of UDP. Now for securing data while data transfer, DTLS should be implemented, because security has been a compulsory feature in WebRTC. SRTP and SCTP are the application level protocols which are being used in WebRTC for multiplexing of various streams, providing congestion and flow control, providing partial reliable data delivery and so on.

In addition to above-mentioned protocols, WebRTC will also use session description protocol (SDP). This protocol provides data format for negotiating parameters of peer to peer connection. Each WebRTC client uses SDP to inform the other client about which transport protocols, ports, codec and other components to use for communication.
CHAPTER 7: PEER TO PEER CONNECTION ESTABLISHMENT

In general, setting up peer to peer connection requires a lot more effort when compared to a regular web browsing model where HTTP handshake mechanism is used for negotiating the parameters of the connection. HTTP mechanism always assumes that the server has public IP address and is reachable by the client, if not client and server may present in the same internal network.

7.1 Network Address Translator

When IPv4 got introduced which is a 32 bit long, everyone has thought that the available IP addresses would be enough for all the internet users around the world. This version of internet protocol can only provide 4.29 billion IP addresses which will not be enough for all the internet devices that are out there. Therefore, available IP addresses have been classified into private and public addresses. Also, a network address translator (NAT) have been introduced at the edge of a network where all the internet devices behind NAT will be assigned one of the private address from the ranges below shown in Figure 5. Whereas, NAT devices will be assigned a public IP address, which has the responsibility of tracking all the internet devices behind it by maintaining a table mapping of IP addresses and ports as shown in Figure 6 below. This gives the flexibility to reuse private IP addresses among many different internet devices behind NAT.

<table>
<thead>
<tr>
<th>IP address range</th>
<th>Number of addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0-10.255.255.255</td>
<td>16,777,216</td>
</tr>
<tr>
<td>172.16.0.0-172.31.255.255</td>
<td>1,048,576</td>
</tr>
<tr>
<td>192.168.0.0-192.168.255.255</td>
<td>65,536</td>
</tr>
</tbody>
</table>

*Figure 5: Reserved Private IP ranges*
When it comes to WebRTC, it is more likely that WebRTC clients can have distinct private IP addresses behind one or more layers of NATs as shown in Figure 7 below. Now, one shouldn’t transfer private IP addresses. Therefore, these peers can’t be reached directly from one another. In order to start a session at first all the conceivable IP addresses and port numbers between peers needs to be gathered for NAT traversal. Once the possible IP list has been built, connectivity checks need to be run for finding the ones that work. Building possible IP list, NAT traversal and connectivity checks are the challenges for establishing peer to peer connection.

Figure 6: IP Network Address Translator

<table>
<thead>
<tr>
<th>Internal</th>
<th></th>
<th>Public</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>Port</td>
<td>IP</td>
<td>Port</td>
</tr>
<tr>
<td>192.168.0.1</td>
<td>1337</td>
<td>50.76.44.114</td>
<td>31454</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7: Peers behind NAT devices
What happens in case of HTTP is that peers assume server will always be available for making new connections and listening to handshakes sent by them. But the same can’t happen in case of remote peer cause the remote client can be offline or unreachable sometimes. Therefore, additional problems should be resolved, in order to make a successful peer connection as mentioned below.

- Initiating peer that wants to open the connection should notify the remote peer to begin listening for packets.
- Next potential routing paths should be identified between peers. Once the routing path gets ready, the same information should be provided between peers for peer to peer connection establishment.
- Once the connection is established, information regarding the parameters of media and data streams needs to be exchanged such as protocols, encoding used and so on.

The good thing is that WebRTC has built-in ICE protocol for performing mandatory routing and connectivity checks. The remaining things for establishing successful peer to peer connection has been left to application developers such as signaling and initial session negotiation.

### 7.2 Signaling & session negotiation

Before start to begin any network check or session transaction, peer has to check whether the remote peer is reachable and willing to set up the connection or not? This mechanism is known as offer and answer where an initiating peer will send an offer to the remote peer. Now the remote peer should send an answer back for a successful peer connection establishment. In order to achieve this offer and answer mechanism, a shared signaling channel is needed between peers to notify each other about the connection.
The choice of protocol being used for signaling transport has been left to application developers making it easy to choose from a variety of available signaling protocols. Also, it enables the interoperability making it possible for WebRTC to communicate with an existing communication infrastructure like SIP, Jingle, ISDN and many more. In such cases signaling server acts like a gateway to communicate with an existing communication network. The responsibility will be on the configured network to notify the target peer for the connection offered and send the answer back to the WebRTC client that initiated the exchange. If required, WebRTC can also choose to have its own particular signaling service with a custom protocol built-in to communicate the messages. For example, a telephone call can be initiated using PSTN client.

7.3 Selecting a Signaling Service

For setting up a signaling service, WebRTC spec allows clients to use any technology for this, from XMP, email to smoke signals. There are many signaling options to choose from in WebRTC. Below is the partial list of popular signaling options for connecting candidates in WebRTC.

- **HTTP POST**
  This will be the better option for strictly web base applications.

- **Google's Channel API**

- **XHR Polling**
https://appr.tc/ uses Google's channel API with XHR polling.

- Server-Sent Events
- XMPP (Jabber) over Websockets
  This will be the best choice for instant messaging apps.
- SIP over WebSockets
  This will be the best option for server to server communication. SIP will give compatibility with a widest range of servers as its standard based and it's been available for a very long time. Using this, we can extend existing SIP app to the web.
- JavaScript SIP Libraries (JsSIP, SIP-JS, SIPML5...)
- WebSockets/JSON
- Jingle (Used by GoogleTalk)
- Signalmaster (it is an open source signaling server)

The choice of signaling option depends on the type of application that is being built and what other system or protocols need to be interface with.

### 7.4 Google Channel API

Consider the case of Google's channel API for this technology case study. This API has been designed for those applications updating information immediately. Using this, any application can send messages across the network by making connection with the Google server. The server responsibility is to create unique channel for individual clients participating in the connection. For this, server will make use of client ID for identifying established channels for individual clients. In case of webRTC, an initiating peer will most likely to have a username or some form of unique id which is known as the Client ID. Also, the server will send a unique token in response to the clients in order to connect and listen to a channel as shown in Figure 9 below. In order to send message across the network, an initiating client will send an HTTP request to the server which identifies the client by client ID and forwards the message to the destination
client using the correct channel. Now, the receiving end performs HTTP post with the server which will process the message and forwards it to the other client over the channel as shown in Figure 10 below.

![Figure 9: Channel Message Request](https://cloud.google.com/appengine/docs/python/channel/)

![Figure 10: Channel Message Response](https://cloud.google.com/appengine/docs/python/channel/)
7.5 Session Description Protocol (SDP)

Once the common signaling channel has been implemented, the next step is to initiate WebRTC connection. Once the media stream is available from the browser, a session description offer (SDP offer) of the initiating peer connection needs to be generated. The generated SDP offer will be sent to remote peer using already implemented signaling service. The remote peer will register its own media stream to generate an answer SDP description, once SDP offer is received. The generated session answer (SDP answer) will be sent back connection initiator.

The session description protocol (SDP) is a text based protocol and is used in WebRTC for describing the parameters of the peer to peer connection. The purpose of using SDP is only to describe session profiles and it will not deliver any media by itself. A session profile can represent a list of properties for a connection establishment, such as types of media type (audio, video, application data), transport network protocols (UDP), settings of codec used, bandwidth information and other Meta data.

Below is sample generated SDP offer.

![Sample SDP offer](image)

| 1 | Secure audio profile with feedback |
| 2 | Candidate IP, port, and protocol for the media stream |
| 3 | Opus codec and basic configuration |

**Figure 11: Sample SDP offer**


With this session negotiation has been completed, but connectivity checks and NAT traversal needs to be performed still.
7.6 Interactive Connectivity Establishment (ICE)

For establishing successful peer to peer connection, participating clients should be able to exchange packets from one another. This is an important step in achieving real-time communication, which may sound simple but hard in practice because of many layers of firewalls and NAT devices between most clients.

In order to explain this easily first let’s assume that both the participating clients are in an internal network. Also, assume that there are no firewalls or NAT devices between them. Now, in order to establish a connection, each client has to request their own operating system for its IP address. Once the IP address is available, it will be appended to the generated SDP string along with the port number and forwarded to the remote client. As soon as the session description exchange gets finished between clients, they should be able to initiate direct peer to peer communication.

The above scenario doesn’t work well unless the public routing path is known when the clients present on distinct private networks. All this complexity is managed by using “ICE agent” of WebRTC framework. An ICE agent is responsible for gathering IP addresses, performing connectivity checks and keeping connection alive.

Once the session description is set on either local or remote client, an ICE agent automatically starts its process for identifying the possible IP address and the port number for each client which are also known as ICE candidates. An ICE agent can perform three different queries for building possible ICE candidate list as described below.
• An ICE agent can query its own operating system for obtaining client IP address.

• If implemented, an ICE agent can query a STUN server from outside for obtaining the public IP address and the port number of the requested client as shown in the Figure 12 below.

• If implemented, an ICE agent appends an external TURN server. Whenever STUN fails, this acts as a relay server for sending packets to and from clients as shown in the Figure 13 below.

![Figure 12: STUN request for public IP and port](https://example.com/figure12.png)


![Figure 13: TURN relay server](https://example.com/figure13.png)


So whenever a new ICE candidate gets discovered, it will be registered automatically and notified by an ICE agent. Once this is done, an ICE checklist will be built using the local and remote ICE candidates as shown in Figure 14 below.
After having the candidate pairs ready, SDP offer will be generated to send to the remote client over the signaling channel. Once the remote client receives ICE candidates, it starts to generate an answer SDP which contains its own list of ICE candidates. Once the full list has built from both the peers, ICE agent will start to perform connectivity check to see if both peers can reach from one another. While performing the connectivity check an ICE agent sends a message to other peer to see if it can acknowledge a successful STUN response. This mechanism is referred to as STUN binding request. If this STUN connectivity check gets completed correctly, then will successfully have a routing path for shared connection as shown in Figure 15 below. Contrarily, if ICE fails, then the connection will fall back to TURN relay server provided it is configured as shown in Figure 16 below.
7.7 STUN binding request

In order to establish peer to peer connection, participating clients behind NAT must know their public IP address. For this, WebRTC makes use of STUN server which is a client server protocol. WebRTC clients perform request response transaction on the STUN server which returns the public IP and port of respective users. Figure 17 shows the format of STUN message header which starts with a fixed header and zero or more attributes. The header portion of the STUN message includes a method, class and transaction
ID. STUN server has support for various methods depending on the type of various incoming requests. WebRTC makes use of binding method in order to determine a particular binding that NAT devices has allocated to STUN clients and keeping these bindings alive. A class in the header indicates whether action is a request, a response or an error. Every transaction with the STUN server includes a transaction ID which helps in associating a successful STUN response with the request generated for it and this transaction ID is a randomly chosen 96-bit number. Optional attributes are included along with the header to specify any additional information regarding a STUN message such as message length or other value extensions.

![Figure 17: Format of STUN Message Header](https://tools.ietf.org/html/rfc5389)

Consider the above STUN configuration as shown in Figure 17, for explaining a successful STUN binding request/response transaction. The configuration includes a STUN client and a STUN server with two NAT devices in between them. Now, when the client initiates a binding request, it has to pass through NAT1 and NAT2 in order to reach to the server. When a request is received by NAT1 device, it will modify the source IP and port to its local IP address and forwards it to NAT2. NAT2 will forward the packet to STUN server by modifying the source transport address to public IP and port created by it. Now, this address is known as reflexive transport address being received by STUN server. Now, the STUN server copies the reflexive transport address into an attribute, which remain untouched as it passes back to the STUN client through the NAT devices. Now, the STUN client will have its public IP and port after a successful STUN binding response.
But the thing is that, STUN alone won’t be enough and TURN server got introduced only to guarantee the connectivity for those peers that fail to establish peer to peer connection. At present, the fall back rate to TURN server is 1 out of 7 calls due to various limitations. An ICE agent will always use the most direct connection possible as it ranks and prioritizes the order of performing connectivity checks. Therefore, a successful STUN server connection will always be preferred over relay connection from a TURN server. Using this technique, the gathering and systematic testing of ICE candidates can take some time which will be optimized using an extension called Trickle ICE. This process let’s testing ICE candidates before the full list is built which speeds up the process. Hence, an ICE framework can efficiently introduce peers or ICE candidates and get them connected even across NAT's and Firewalls.
7.8 Options for Server Setup/Hosting

Below are some of the third party server side solutions.

- Tokbox.com Open Talk Platform: This provides a robust signaling solution along with their open source open talk WebRTC API.
- Xirsys.com hosted server infrastructure: Xirsys offers signaling as part of their hosted server infrastructure for WebRTC and they also provide some WebRTC components to use along with their service.
- OnSIP.com/WebRTC hosted PSTN gateway: OnSIP offers PSTN gateway and they are globally interoperable with SIP end points.

There are many options available for STUN server deployment as shown below.

- The below link provides a list of freely available public STUN servers.
  (https://gist.github.com/yetithefoot/7592580)
  Or
  Users can create own STUN server using Node.js

- Users can also use open source STUN server software mentioned below.
  (http://stunprotocol.org/)

Below are the couple of options for TURN server.

- RFC5766 project which can be found on Google Code.
- NUMB is an open and free hosted STUN and TURN servers.

At this point peers have completed offer-answer workflow, NAT traversal and connectivity checks which means that they have raw UDP connections available to one another. Now, below chapter explains about what happens after a successful connection establishment.
CHAPTER 8: DELIVERING MEDIA AND APPLICATION DATA

WebRTC protocol stack implements below listed protocols on top of UDP for performing several additional mechanisms such as congestion control, flow control, bandwidth management, and many more for better performance. Moreover, UDP transfers data in clear format which must be encrypted for transmission. Hence, below mentioned protocols must be implemented on top of UDP to meet WebRTC requirements.

- Datagram Transport Layer Security (DTLS)
  DTLS provides secure transport of information by negotiating secret keys for media encryption.
- Secure Real-Time Transport (SRTP)
  SRTP used for transporting media data such as voice and video.
- Stream Control Transport Protocol (SCTP)
  SCTP is utilized for transporting arbitrary data such as texts, files, images.

8.1 Secure Communication with DTLS

WebRTC specification has made encryption mandatory for data (audio, video, application data) being transferred over UDP between peers. For this, transport layer security protocol should work well except for the fact that it can’t be implemented on top of UDP because it depends on TCP which provides in-order and reliable data transport. Hence, WebRTC uses DTLS on top of UDP which provides security features similar to TLS.

Intentionally, DTLS has been designed similar to TLS with minimum modifications to make it work on UDP. Below are the problems addressed by UDP when compared with TLS.

- For negotiating the tunnel, TLS requires in-order and reliable transport of handshake records.
- Records shouldn’t be fragmented, because it may result in TLS integrity check failure.
- Records shouldn’t be transported out of order if so TLS integrity check may fail.
In case of TLS, each record should process, as it is determined by the handshake algorithm. Also, some records can have multiple packets. Therefore, DTLS implements Mini-TCP for handshake process sequence.

![Sample DTLS handshake](image)

*Figure 19: Sample DTLS handshake*


A fragment offset and sequence number will be added for every DTLS handshake record, in order to address the orderly delivery requirement of TLS. More over by doing this, huge data can be fragmented into packets and can be gathered on the other end. Hence the records can be transmitted in the order determined by this protocol. Finally, DTLS should also deal with packet loss. A simple timer will be used for retransmitting those handshake records which are not received with in an expected interval.

Hence the combination of above three (Fragment offset, record sequence number and retransmission) permits DTLS to perform the handshake over UDP. Now for completing this sequence, peers in the network should generate self-signed certificate for following regular TLS handshake protocol.

In case of WebRTC client, self-signed certificates will be generated automatically for every peer in the network, allowing not to use any certificate chain to verify. These self-signed certificates will be used for performing DTLS handshake between WebRTC clients and can’t be used for peer authentication as there is no explicit chain of trust. Encryption and integrity will be provided by DTLS but implementing authentication has been left to the application developer. Identity and authentication are the important
security mechanisms and must be implemented for all the participating peers in a WebRTC session. Below are the two suggested ways for providing authentication and identity verification.

- Before setting up a WebRTC session, web application can utilize its existing identity verification system (Eg: User log-in credentials) for performing user authentication.

- Another way is to specify an identity provider for each participating peer while generating SDP offer/answer. When this generated SDP messages received by other ends, each peer can contact their own identity provider for performing the verification of received certificates.

Now with the completion of encryption, integrity, identity, authentication, DTLS handshake prerequisites has been met. DTLS adds two important rules for processing of unordered records. These rules must be implemented because UDP doesn’t handle fragmentation and reassembly when compared to TCP.

- DTLS data should fit into a one network packet only.
- DTLS should use block cipher for implementing data encryption.

8.2 Delivering media with SRTP&SRTCP

Using the WebRTC API, media constraints will be specified in application to acquire media streams, and then registering them with RTCPeerConnection object. Starting from media acquisition to delivery, a lot of things needs to be taken care of such as encoding optimization, handling packet loss, network fluctuations, error recovery, flow control, and many more. All of this will be taken care by WebRTC media and network engines provided by browser application. The browser network engine utilizes its own stream control algorithm which permits every connection to start streaming at a bit rate of less than 500 Kbps, regardless the size and quality of provided media stream. From there, the network stack will start to dynamically adjust the media stream quality depending on the available network bandwidth. This process is continuous and it occurs throughout the life span of the connection to overcome packet loss and insufficient bandwidth situations. Therefore, it is the responsibility of the WebRTC network engine to
attempt to adjust provided media streams to coordinate with the current states of the system. For this WebRTC uses existing transport protocols to improve and adjust the quality of every media stream as mentioned below.

- Secure Real-time Transport Protocol (SRTP): This is the secure profile of the standardized packet format for delivery of real time data over the IP network.
- Secure Real-time Control Transport Protocol (SRTCP): This is the secure profile of the control protocol for delivering client statistics and control data for establishing SRTP flow.

WebRTC uses the secure profile of RTP protocol because it requires information to be encrypted throughout the communication process. Below Figure 20 shows the standard packet format of SRTP for delivering media over IP networks.

<table>
<thead>
<tr>
<th>Bit</th>
<th>+0..7</th>
<th>+8..15</th>
<th>+16..23</th>
<th>+24..31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>V</td>
<td>P</td>
<td>XCC</td>
<td>M</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td>Sequence number</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timestamp</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td>Synchronization source (SSRC) identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+32</td>
<td>Contributing source (CSRC) identifier (optional)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+32</td>
<td>+32</td>
<td>RTP extension (optional)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+N</td>
<td></td>
<td>Encrypted RTP payload ...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td>SRTP MD5 (optional) + Authentication Tag (optional)</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 20: SRTP header*


- An auto increment sequence will be carried by SRTP packet to enable the receiver for detecting out of order delivery of media data.
- A timestamp will be carried by SRTP packet for synchronization of various media stream tracks.
- An SSRC identifier is a unique stream ID, carried by SRTP packet used for associating each incoming packet with an individual media stream.
- An encoded media payload and authentication tag are included in the packet for verifying the integrity of delivered packet.
An SRTP packet contains all the crucial data needed by the media engine to playback real-time media of the stream. Anyhow, it is the duty of SRTCP protocol to control delivery of respective SRTP packets. For this, SRTCP protocol utilizes an alternate out of band feedback channel for every media stream. SRTCP keeps track of SRTP packet statistics such as the number of packets sent and lost, latest received packet sequence number, monitoring timestamp and so on.

WebRTC has special requirements for these protocols to make them compatible with the real time communication standards, as described below.

- As described earlier both SRTP and SRTCP provides encrypted pay load data for communication which is secure, but neither of these protocols provide a mechanism for sharing secret keys which will be used for message authentication in further process. This is the reason for protocol stack showing DTLS handshake on top of UDP even before SRTP and SRTCP. Hence, DTLS handshake needs to be performed in the beginning which in turn shares a secret key between the peers. This shared key can be reused in later stages as keying material within SRTP and SRTCP.

- For every individual stream, SRTP and SRTCP requires different ports which will cause trouble for those clients behind firewalls and NAT's. Therefore, WebRTC requires an additional multiplexing extension which can deliver multiple incoming streams coming through various ports on the same destination port.

In a nutshell, these two protocols (SRTP, SRTCP) along with DTLS can work together on top of UDP for optimizing the real time delivery of media streams (Audio, Video). The application developers do not have to manage these protocols directly because the major part of the required infrastructure has already been implemented within the browser.
8.3 Delivering application data with SCTP

WebRTC makes use of DataChannel API for implementing peer to peer arbitrary application data transfer. Earlier discussed SRTP protocol is only suitable for media transfer, and it doesn’t support transporting application data. Hence, this API depends on steam control transmission protocol (SCTP) for arbitrary application data transfer.

Below are the WebRTC requirements for implementing DataChannel API.

- Multiplexing of numerous independent channels should be supported by transport protocol.
  - Every channel should support either in order or out of order delivery.
  - Every channel should support either reliable or unreliable delivery.
  - A channel can have a preference level as determined by application.
- If required, application should allow messages to be fragmented and restored by data transport.
- Flow control and congestion control mechanisms should be implemented.
- Integrity and confidentiality of data should be maintained while doing the transport.

Reading through the requirements, one can recognize that WebRTC requires both UDP (unreliable, out of order delivery) and TCP (reliable, in order delivery, message fragmentation, and channel priority support and multiplexing) behaviors combined. This is the reason SCTP is recommended for data transport which gives the best elements of both the worlds TCP and UDP as shown in the Figure 21 below. However, SCTP can run directly on top of IP protocol as similar to TCP and UDP. Whereas, in case of WebRTC, SCTP is tunneled over DTLS, which runs on top of UDP. The good thing here is that DTLS satisfies a couple of WebRTC data transport requirements such as data encryption within payload, integrity and confidentiality.
Figure 21: Comparing TCP vs. UDP vs. SCTP

Below Figure 22 shows the standard packet format of SCTP for delivering arbitrary data over IP networks.

Figure 22: SCTP header and data chunk

An SCTP packet is a combination of common header and data or control chunk. The SCTP header will contain the source and destination ports, an arbitrary-produced confirmation tag and checksum for the entire packet. A header will be followed by one or more data chunks as shown in above Figure 22. Type indicates the data type of all the chunks. U, B and E bits are utilized to determine the order and positioning of messages over various data chunks. U bit indicates that the data chunk is unordered. B and E bits are utilized in determining the starting and end of fragmented messages as shown below.
<table>
<thead>
<tr>
<th>B</th>
<th>E</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>The first fragment of a message</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Middle fragment of the message</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Last fragment of message</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Unfragmented message</td>
</tr>
</tbody>
</table>

Size of the data chunk will be determined by length bits. In order to detect duplicate deliveries on the receiver end, a 32-bit transmission sequence number (TSN) will be used internally by SCTP which will acknowledge the packet receipts. Unique stream identifier bits are utilized for associating individual data chunks with active available streams which is also known as stream multiplexing. An auto increment message number known as, stream sequence number will be used for sequencing the fragmented messages with respect to its associated stream. SCTP will continue to use the stream sequence number, even if the unordered bits are set but individual messages will be delivered in out of order fashion. Any additional metadata regarding the transferred chunk will be filled in payload protocol identifier bits (PPID). In total, each SCTP data chunk will have an overhead size of 28 bytes consisting of 12 bytes for the common header and 16 bytes for data chunk header followed by application payload.

Now, in order to establish SCTP connection, it implements TCP similar handshake to negotiate the starting parameters for association. Below are specific WebRTC requirements with respect to SCTP functionality.
• SCTP provides mechanism for delivering unordered messages but it doesn’t explicitly provide any guaranty of delivery which crucial feature for implementing WebRTC data channel. For this, WebRTC clients should implement the partial reliability extension which in turn extends SCTP protocol for allowing custom delivery guarantees.

• SCTP doesn’t offer any fields within its protocol for prioritizing individual streams which is a WebRTC requirement. Hence, this feature should be executed higher in the WebRTC stack.
CHAPTER 9: DATACHANNEL

DataChannel API enables associated peers to open one or more channels in order to transfer arbitrary application data between peers. This API is similar to WebSockets, but it offers additional features with customizable and reliable delivery properties for underlying flexible transport protocols and data can be transferred peer to peer. Everything is fully secure as it uses standard DTLS encryption to make sure the packets that got sent across the data channel are fully encrypted on their way to destination.

Even before transferring application data, peers have to negotiate the SCTP parameters in order to implement a successful connection. Hence, the participating peers has to advertise the SCTP parameters while performing SDP offer-answer workflow. Below is sample SDP snippet for SCTP association.

![Sample SDP snippet for SCTP association](image)


Now peers should be able to exchange arbitrary data once channel parameters are communicated.
CHAPTER 10: MULTIPARTY ARCHITECTURE

A single WebRTC peer to peer connection may easily consume significant bandwidth for transporting HD media streams. Hence, WebRTC architecture needs to be planned carefully while implementing multi-party conference calls where individual streams need to be aggregated and distributed among peers. Below Figure 24 shows the distributed architecture of multi-party call in WebRTC.

Implementing, one-to-one connections are easy as it involves a single direct connection between peers which doesn’t require any further network optimization. In case of multiple party conference, each peer will try to establish a connection with every other participating peer in the WebRTC call which results in a mesh network. This solution is bandwidth intensive with everyone on the call handling multiple incoming streams and their own outgoing stream. So here network may have to limit video quality or limit participants. In general, broadband connections don’t work well with this type of topology. In order to overcome the problem of a mesh network, star topology has come into the picture where every individual peer connects to a super node, which takes the responsibility of distributing streams all across the network. The option of choosing a super node has been left to application developers and it can be any other peer in
the network or a dedicated service can be built for data distribution. In a simple case, initiating peer can act as a super node. For a better performance, a developer can establish a mechanism to pick a peer with the best available bandwidth. For the most robust conference architecture MCU (Multi point control unit) is recommended. This is a server that's custom made for relaying large amounts of audio and video. It can do various things such as selective stream forwarding and recording. So if one peer drops out, it won't interrupt the whole conference because the MCU is taking care of everything.
CHAPTER 11: SECURITY IN WebRTC

- WebRTC has built-in security features as it is built right into fully protected browser and is not a plug-in.
- Security updates will happen when the browser is updated. This prevents malware being installed from innocent looking plug-in.
- Encryption is mandatory in all WebRTC components including signaling. So all the data being sent by WebRTC is encrypted using standard AES encryption. Unencrypted data can't be intercepted between browsers.

In order to take full advantage of security, WebRTC clients should implement protocols as shown in above Figure 25.

- WebRTC should use secure protocols such as HTTPS for doing signaling.
  The data will be fully secured using the protocols Secure Real Time Transport Protocol (SRTP) for media and Datagram Transport Layer Security (DTLS) for data channel.
- Explicit access dialogue box is compulsory while accessing microphone and camera
CHAPTER 12: USE CASES AND INSPIRATION

- File sharing
  https://sharefest.me/
  It allows peers to directly share files without uploading them to cloud.

- Screen Sharing
  Currently using in Google Hangout

- Video conferencing / Telepresence
  Startups trying to get rid of Skype in the space by connecting browsers with mobile networks and plain old telephone networks, (PSTN)

- Integrated text chat
  It's old but still now it's available peer to peer between browsers.

- Virtual Conferences
  Possible by mixing the multimedia capabilities of HTML5 with video, audio and data.

- VOIP calls
  Not just Vonage or Google hangouts
  https://zingaya.com/

- Remote Desktop Applications

- Multiplayer games

  Now by integrating other API's into WebRTC

  - Facial Recognition
  - Image Capture
  - Video Record and Downloading
Live examples:

- https://appr.tc/
  Quick video conference and the code is also available open source

- http://www.addlive.com/

- https://vline.com/
  Free video calls and Chat rooms

- https://vsee.com/
  Conferencing and screen sharing application.

Debugging tool for WebRTC developers

- chrome://webrtc-internals/
  It shows all statistics about what's actually happening inside a call such as packet loss, bandwidth, video resolution and sizes. Also, there will be a full log of all the calls made to the WebRTC API that users can download and export.
CHAPTER 13: CONCLUSIONS

Internet and World Wide Web together revolutionized the way we interact with machines and humans starting from accessing information to communicating other parts of the world. Now, with the introduction of WebRTC, browsers can do interactive audio and video without having to install any third party software or plug-in. This technology case study is about how existing protocols got combined into an overall architecture for providing peer to peer communication. The major advantage about implementing WebRTC is that, it is not just limited for browser communication. WebRTC technology will add many new user interfaces for websites by installing gateways, in order to interact with existing communication technologies.

13.1 RECOMMENDATIONS

It is recommended that while implementing WebRTC, the developer has to mainly take care of services such as setting up signaling service, Traversing firewalls and NAT devices, data distribution and efficiency, finally delivery and reliability. It is recommended to use low latency signaling transport for better performance. While performing NAT traversal, it is recommended to use trickle ICE whenever possible which results in setting up faster connections and more signaling. In case of a multi-party conference, it is recommended to use a dedicated service as a super node and also data optimization needs to be done before forwarding it to other peers in the network. Data channel should be implemented in such a way that it should be able to adjust to changes in the states of a system and should provide bandwidth management. Prefer using unorderly delivery to stay away from head-of-line blocking. If at all orderly delivery used, then transmitting packet size should be minimized which will diminish the effect of head-of-line blocking and packet loss.
REFERENCES


