NTNU – Trondheim
Norwegian University of
Science and Technology

Unified Communication and WebRTC

Xiao Chen

Submission date: May 2014
Responsible professor: Mazen Malek Shiaa, ITEM
Supervisor: Mazen Malek Shiaa, ITEM

Norwegian University of Science and Technology
Department of Telematics

Title: Unified Communication and WebRTC

Student: Xiao Chen

Problem description:

Web Real-Time Communication (WebRTC) offers application developers the ability to write rich, real-time multimedia application (e.g. video chat) on the web, without requiring any plugins, downloads or installations. WebRTC is also currently the only existing soon-to-be standardized technology on the market to create horizontal cross-platform communication services, encompassing smartphones, tablets, PCs, laptops and TVs, which adds value for both consumers and enterprises. WebRTC gives operators the opportunity to offer telephony services to more devices, such as PCs, tablets and TVs. This thesis considers how WebRTC can enhance the existing echo-systems for telephony and messaging services by providing the end-user rich application client.

It will also covers research about different solutions to implement WebRTC to cooperate with existing telephony services like hosted virtual Private Branch Exchange (PBX) services.

A prototype of WebRTC deployment based on different rich communication scenarios will be implemented along with this thesis. Some corresponding test and evaluation will be fulfilled in this prototype.

Research about advanced WebRTC usability in telephony and messaging services will be covered in this thesis by the feedback of the WebRTC prototype

Responsible professor: Mazen Malek Shiaa, ITEM

Supervisor: Mazen Malek Shiaa, ITEM

Abstract

For the development of traditional telephony echo-systems, the cost of maintenance traditional telephony network is getting higher and higher but the number of customer does not grow rapidly any more since almost every one has a phone to access the traditional telephony network. WebRTC is an Application Programming Interface (API) definition drafted by the World Wide Web Consortium (W3C) that supports browser-to-browser applications for voice calling, video chat, and Peer-To-Peer (P2P) file sharing without plugins.[Wik14s] "This technology, along with other advances in HyperText Markup Language 5 (HTML5) browsers, has the potential to revolutionize the way we all communicate, in both personal and business spheres."[JB13a]

As network operators aspect, WebRTC provides many opportunities to the future telecommunication business module. For the users already have mobile service, operator can offer WebRTC service with session-based charging to the existing service plans. Messaging APIs can augment WebRTC web application with Rich Communication Services (RCS) and other messaging services developers already know and implement. Furthermore, since WebRTC is a web based API, then the implementation of Quality of Service (QoS) for WebRTC can provide assurance to users and priority services (enterprise, emergency, law enforcement, eHealth) that a WebRTC service will work as well as they need it to. WebRTC almost provide network operator a complete new business market with a huge amount of end-users.

As an end-user aspect, WebRTC provides a much simpler way to have real-time conversation with another end-user. It is based on browser and internet which almost personal or enterprise computer already have, without any installation and plugins, end-user can have exactly the same service which previous stand-alone desktop client provides. By the system this thesis will cover, the end-user even can have the real-time rich communication service with multiple kinds of end-users.

This thesis will cover the research about how to apply WebRTC technology with existing legacy Voice over Internet Protocol (VoIP) network.

Keywords : WebRTC, AngularJs, Nodejs, SIP, WebSocket, Dialogic XMS

Preface

WebRTC is quite popular topic in the web development filed since the massive usage and development of HTML5 web application on the internet. The initial purpose of this web API is to provide the browser client the ability to create real-time conversation between each other. After many WebRTC based application come out the market, it is quite normal to think about how to integrate these kind of web application with the current legacy telephony network as the next big step for this technology. The requirement of this process is not only from the traditional telephony operator but also the normal end-users. The approach to achieve this goal is the man purpose of this thesis.

Research about current WebRTC technology usage and development of a WebRTC prototype system are the two main parts of this thesis. The prototype system is implemented by regarding to the research of WebRTC integrated with legacy telephony network.

Current status of WebRTC technology, WebRTC business use cases, analysis of different possible WebRTC implement solutions and WebRTC system architecture will be covered in this thesis. Some research regarding with the development of WebRTC prototype system will be covered in this thesis as well.

The prototype described in this thesis is implemented to cooperate with existing legacy VoIP network services through Session Initiation Protocol (SIP) server and PBX[1] service. It will provide most of essential functions which are included in the legacy telephony business, besides other communication functions used on web. Moreover, some analysis and discussion about the feedback of the prototype will be covered in this thesis.

The prototype will be implemented in programming language Javascript for both client font-end and server back-end by using the AngularJs framework and Nodejs framework mainly. The approach and reason to choose these framework and programming language will be expounded in the later chapter in this thesis.

[1] Users of the PBX share a certain number of outside lines for making telephone calls external to the PBX.[Web14b]

Acknowledgment

Written by Xiao Chen in Trondheim in May 2014

Thanks for Mazen Malek Shiaa, ITEM

Frank Mbaabu Kiriinya, Gintel AS

Roman Stobnicki, Dialogic, the Network Fuel company

Special thanks for Gintel AS

Contents

List of Figures

List of Tables

List of Code Snippets

List of Acronyms

API Application Programming Interface.

DTLS Datagram Transport Layer Security.

GIPS Global IP Solutions.

HTML5 HyperText Markup Language 5.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol over Secure Socket Layer.

ICE Interactive Connectivity Establishment.

IETF Internet Engineering Task Force.

IP Internet Protocol.

JSON JavaScript Object Notation.

MVC Model–View–Controller.

NAT Network Address Translator.

P2P Peer-To-Peer.

PBX Private Branch Exchange.

PHP PHP: Hypertext Preprocessor.

PSTN Public Switched Telephone Network.

QoS Quality of Service.

RCS Rich Communication Services.

RTC Real-Time Communication.

RTP Real-time Transport Protocol.

SDP Session Description Protocol.

SIP Session Initiation Protocol.

SRTP Secure Real-time Transport Protocol.

STUN Session Traversal Utilities for NAT.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

TOR The Onion Router.

TURN Traversal Using Relays around NAT.

UA User Agent.

UAC User Agent Client.

UAS User Agent Server.

UDP User Datagram Protocol.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

VoIP Voice over Internet Protocol.

W3C World Wide Web Consortium.

WebRTC Web Real-Time Communication.

XMPP Extensible Messaging and Presence Protocol.

Chapter 1
Introduction

In this Chapter, introduction of WebRTC and SIP network will be covered. SIP is one of the VoIP signaling protocols widely used in current internet telephony service which is also the target telephony network integrated with WebRTC application system in this thesis.

1.1 WebRTC

Gmail[1] video chat became popular in 2008, and in 2011 Google introduced Hangouts[2], which use the Google Talk service (as does Gmail). Google bought Global IP Solutions (GIPS), a company which had developed many components required for Real-Time Communication (RTC), such as codecs and echo cancellation techniques. Google open sourced the technologies developed by GIPS and engaged with relevant standards bodies at the Internet Engineering Task Force (IETF) and W3C to ensure industry consensus. In May 2011, Ericsson built the first implementation of WebRTC.

1.1.1 What is WebRTC ?

WebRTC is an industry and standards effort to put real-time communications capabilities into all browsers and make these capabilities accessible to web developers via standard HTML5 tags and JavaScript APIs. For example, consider functionality similar to that offered by Skype[3]. but without having to install any software or plug-ins. For a website or web application to work regardless of which browser is used, standards are required. Also, standards are required so that browsers can

[1] Gmail is a free , advertising-supported email service provided by Google.

[2] Google Hangouts is an instant messaging and video chat platform developed by Google, which launched on May 15, 2013 during the keynote of its I/O development conference. It replaces three messaging products that Google had implemented concurrently within its services, including Talk, Google+ Messenger, and Hangouts, a video chat system present within Google+.

[3] Skype is a freemium voice-over-IP service and instant messaging client, currently developed by the Microsoft Skype Division.[Wik14q]

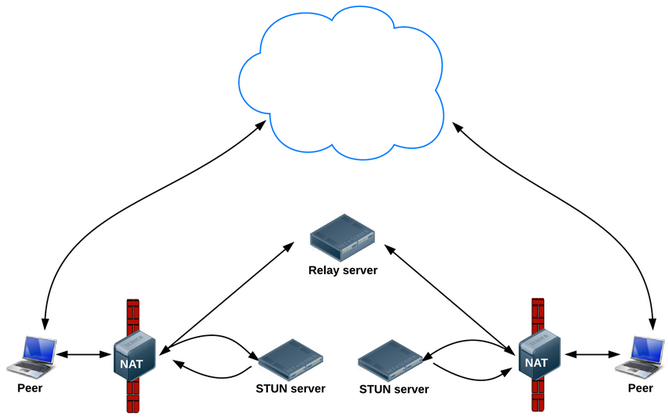

Figure 1.1: WebRTC Network: Finding connection candidates

communicate with non-browsers, including enterprise and service provider telephony and communications equipment[JB13d].

With the rapidly development of internet, more and more communication traffic is moving to web from the traditional telephony network. And in the recent decade, VoIP network services are growing to the peek of the market capacity. Solution to integrate WebRTC and existing VoIP network is the right approach the trend of the internet communication requirement.

1.1.2 WebRTC Network Structure

In the Figure1.1[Dut14] showing how the Interactive Connectivity Establishment (ICE) framework[4] to find peer candidate through Session Traversal Utilities for NAT (STUN) server and its extension Traversal Using Relays around NAT (TURN) server.

Initially, ICE tries to connect peers directly, with the lowest possible latency, via User Datagram Protocol (UDP). In this process, STUN servers have a single task: to enable a peer behind a Network Address Translator (NAT) to find out its public address and port. If UDP fails, ICE tries Transmission Control Protocol (TCP): first Hypertext Transfer Protocol (HTTP), then Hypertext Transfer Protocol over Secure Socket Layer (HTTPS). If direct connection fails—in particular, because of enterprise NAT traversal and firewalls—ICE uses an intermediary (relay) TURN server. In other words, ICE will first use STUN with UDP to directly connect peers and, if that fails, will fall back to a TURN relay server. The expression 'finding candidates' refers to the process of finding network interfaces and ports.[Dut14]

[4]ICE is a framework for connecting peers, such as two video chat clients.[Wik14h]

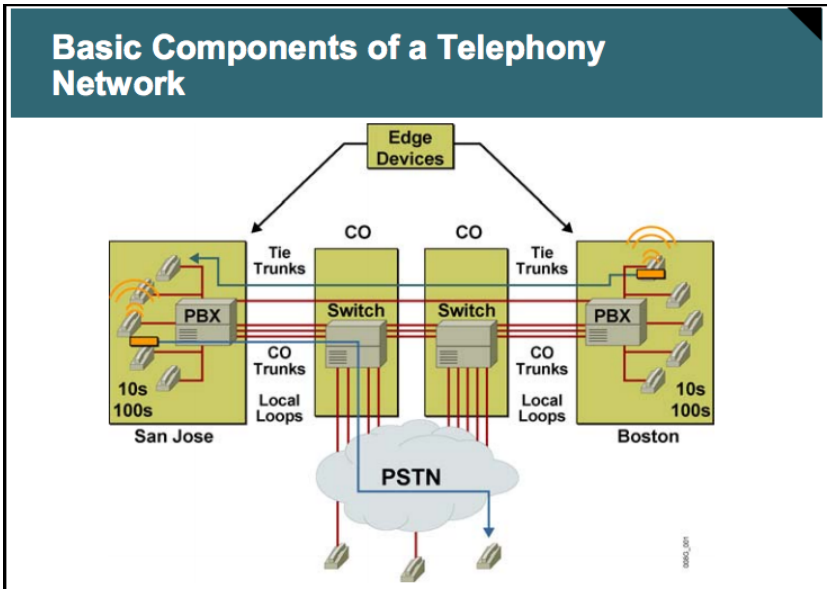

Figure 1.2: Traditional Telephony Network

The difference and usage of STUN server and TURN server will be discussed more detail in Chapter 3.

WebRTC needs server to help users discover each other and exchange 'real world' details such as names. Then WebRTC client applications (peers) exchange network information. After that, peers exchange data about media such as video format and resolution. Finally, WebRTC client applications can traverse NAT gateways and firewalls.

Compare to the traditional telephony network which is shown in Figure1.2[Inc05], the main difference between these two communication network is that WebRTC is P2P communication in STUN server scenario, after the signaling between end-peers, the media data are exchanged directly between tow peers. However, in the traditional telephony, all the media data are transferred to PBX and switches regarding to Public Switched Telephone Network (PSTN)[5] then reach the other side of the peer. Even in TURN server scenario for WebRTC, the media stream is only relaying to the TURN then directly transfer to another peer, no switches involved. Two server working scenario will be discussed in Chapter 2.

[5]The PSTN consists of telephone lines, fiber optic cables, microwave transmission links, cellular networks, communications satellites, and undersea telephone cables, all interconnected by switching centers, thus allowing any telephone in the world to communicate with any other. Originally a network of fixed-line analog telephone systems, the PSTN is now almost entirely digital in its core network and includes mobile and other networks, as well as fixed telephones.[Wik14l]

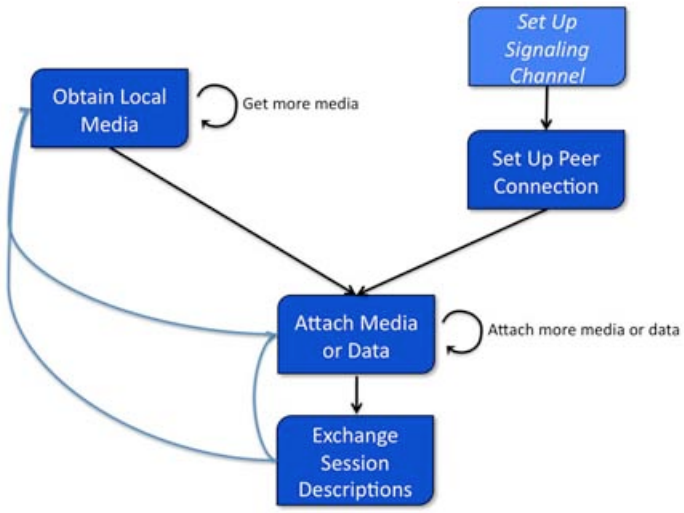

Figure 1.3: WebRTC API View with Signaling[JB13b]

1.1.3 WebRTC Implementation Steps

There are four main steps to implement a WebRTC session shown in Figure 1.3. The browser client need to obtain local media first, then set up a connection between the browser and the other peer through some signaling, after that attach the media and data channels to the connection, afterwards exchange the session description from each other. Finally the media stream will automatically exchange through the real-time peer to peer media channel.

Each step shown in the Figure 1.3 is implemented by some WebRTC APIs. More detail about how to use WebRTC APIs to implement these steps will be covered in Chapter 2. The WebRTC architecture is shown in Figure 1.4, the main focus in this thesis will be Web API part and transport part because Web API is the tool to implement the WebRTC application and transport part is the key for WebRTC application to communicate with application server, media server and other end peer in the system.

Besides WebRTC APIs, signaling is the other important factor in the system. WebRTC uses *RTCPeerConnection* (more about this API will be discussed in Chapter 2) to communicate streaming data between browsers, but also needs a mechanism to coordinate communication and to send control messages, a process known as signaling. Signaling methods and protocols are not specified by WebRTC by Google purpose, so signaling is not part of the *RTCPeerConnection* API.

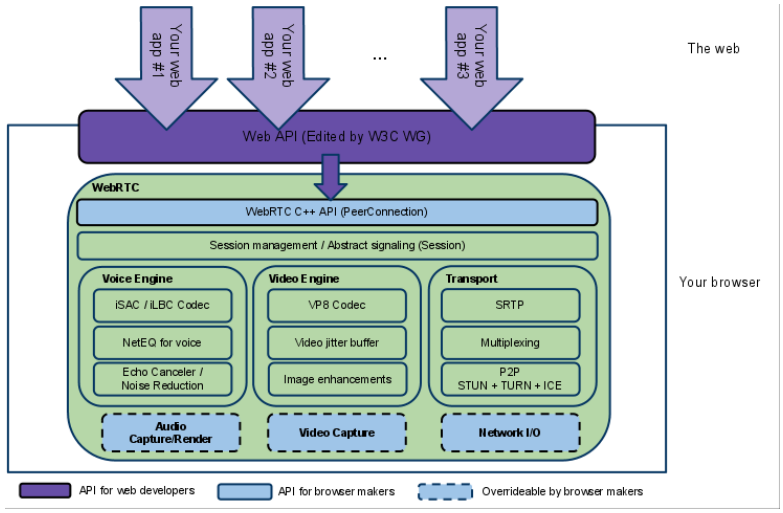

Figure 1.4: WebRTC architecture [Goo12]

Instead, WebRTC app developers can choose whatever messaging protocol they prefer, such as SIP or Extensible Messaging and Presence Protocol (XMPP), and any appropriate duplex (two-way) communication channel. The prototype application in this thesis will use WebSocket[6] as signaling between WebRTC browser end point and keep use SIP as signaling for SIP end point (mobile/fixed phone based on PSTN in this case).

Signaling is used to exchange three types of information[Dut14]:

- Session control messages: to initialize or close communication and report errors.
- Network configuration: to the outside world, the computer's IP address and port.
- Media capabilities: the codecs and resolutions can be handled by the browser and the browser it wants to communicate with.

The exchange of information via signaling must have completed successfully before peer-to-peer streaming can begin. For the prototype application in this thesis, the signaling has two mechanisms, one is for WebRTC browser clients and the other is for SIP clients, it will be explained in Chapter 2.

1.2 SIP

The prototype application in this thesis will be integrated with PSTN through SIP server. Therefore the application server implemented in this system will use SIP

[6]WebSocket is a protocol providing full-duplex communications channels over a single TCP connection.[Wik14t]

signaling to communicate with SIP server to handle the signaling configuration with mobile/fixed phone end-point.

1.2.1 What is SIP?

The SIP is a signaling communications protocol, widely used for controlling multimedia communication sessions such as voice and video calls over Internet Protocol (IP) networks.

The protocol defines the messages that are sent between endpoints which govern establishment, termination and other essential elements of a call. SIP can be used for creating, modifying and terminating sessions consisting of one or several media streams. SIP can be used for two-party (unicast) or multiparty (multicast) sessions. Other SIP applications include video conferencing, streaming multimedia distribution, instant messaging, presence information, file transfer, fax over IP and online games.[Wik14p]

SIP works in conjunction with several other application layer protocols that identify and carry the session media. Media identification and negotiation is achieved with the SDP. It is different key filed format than the WebRTC SDP. For the transmission of media streams (voice, video) SDP typically employs the Real-time Transport Protocol (RTP) or Secure Real-time Transport Protocol (SRTP). For secure transmissions of SIP messages, the protocol may be encrypted with Transport Layer Security (TLS).

1.2.2 SIP Network Elements

In normal SIP network, SIP defines user-agents as well as several types of server network elements. Two SIP endpoints can communicate without any intervening SIP infrastructure. However, this approach is often impractical for a public service, which needs directory services to locate available nodes on the network. In the system implemented of this thesis, the application server will play as 'User Agent', 'Registrar' and 'Gateway' elements in the network.

User Agent[Wik14p]:

A SIP User Agent (UA) is a logical network end-point used to create or receive SIP messages and thereby manage a SIP session. A SIP UA can perform the role of a User Agent Client (UAC), which sends SIP requests, and the User Agent Server (UAS), which receives the requests and returns a SIP response. These roles of UAC and UAS only last for the duration of a SIP transaction.

Registrar[Wik14p]:

A registrar is a SIP endpoint that accepts REGISTER requests and places the information it receives in those requests into a location service for the domain it handles. The location service links one or more IP addresses to the SIP Uniform Resource Identifier (URI) of the registering agent. The URI uses the sip: scheme, although other protocol schemes are possible, such as tel:. More than one user agent can register at the same URI, with the result that all registered user agents receive the calls to the URI.

Gateway[Wik14p]:

Gateways can be used to interface a SIP network to other networks, such as the PSTN, which use different protocols or technologies. In the prototype application, the application server is the gateway to interface a WebRTC WebSocket network (The working process will be covered in Chapter 2).

1.2.3 SIP messages

Since the application server in this system will be used as SIP UA and SIP Gateway, it will send SIP message request to SIP server and receive SIP message request from the SIP server.

One of the wonderful things about SIP is that it is a text-based protocol modeled on the request/response model used in HTTP. This makes it easy to debug because the messages are easy to construct and easy to see. Contrasted with H.323[7], SIP is an exceedingly simple protocol. Nevertheless, it has enough powerful features to model the behavior of a very complex traditional telephone PBX.[Wor04]

There are two different types of SIP messages: requests and responses. The first line of a request has a method, defining the nature of the request, and a Request-URI, indicating where the request should be sent.The first line of a response has a response code.

For sip requests, regarding to RFC 3261[Soc02], the application server in the system will use following SIP messages:

- **REGISTER:** Used by a UA to indicate its current IP address and the Uniform Resource Locator (URL)s for which it would like to receive calls.
- **INVITE:** Used to establish a media session between user agents.
- **ACK:** Confirms reliable message exchanges.
- **CANCEL:** Terminates a pending request.

[7] H.323 is a recommendation from the ITU Telecommunication Standardization Sector (ITU-T) that defines the protocols to provide audio-visual communication sessions on any packet network. The H.323 standard addresses call signaling and control, multimedia transport and control, and bandwidth control for point-to-point and multi-point conferences.[Wik14g]

– **BYE:** Terminates a session between two users in a conference.

The SIP response types defined in RFC 3261 will be listened by application server in the following response codes[Wik14j]:

– **100 Trying:** Extended search being performed may take a significant time so a forking proxy must send a 100 Trying response.
– **180 Ringing:** Destination user agent received INVITE, and is alerting user of call.
– **200 OK:** Indicates the request was successful.
– **400 Bad Request:** The request could not be understood due to malformed syntax.
– **401 Unauthorized:** The request requires user authentication. This response is issued by UASs and registrars.
– **408 Request Timeout:** Couldn't find the user in time. The server could not produce a response within a suitable amount of time, for example, if it could not determine the location of the user in time. The client MAY repeat the request without modifications at any later time.
– **480 Temporarily Unavailable:** Callee currently unavailable.
– **486 Busy Here:** Callee is busy.

By listening these SIP response, the application will send request to either WebRTC browser client or SIP client to play as the gateway role in the system. This gateway mechanism will be introduced in Chapter 2.

1.3 Prototype System Working Flow

The main purpose of this thesis is to make unified communication solution with WebRTC technology.

To connect with the traditional telephony network, the VoIP system bridges the PSTN and the IP network. VoIP systems employ session control and signaling protocols to control the signaling, set-up, and tear-down of calls. They transport audio streams over IP networks using special media delivery protocols that encode voice, audio, video with audio codecs, and video codecs as Digital audio by streaming media. In this prototype, SIP signaling is used because of its widely usage and current target PSTN has SIP server support.

The Figure 1.5 shows the basic working flow of the prototype system. The Web Server is the application server in the system, it mainly bridges the WebRTC browser client with other WebRTC clients and the SIP network. The SIP server bridges the SIP network and PSTN network or traditional telephony network. And also the Media Relay server relay all the media stream from different end clients, in the prototype system, it is a media server provided by Dialogic, the Network

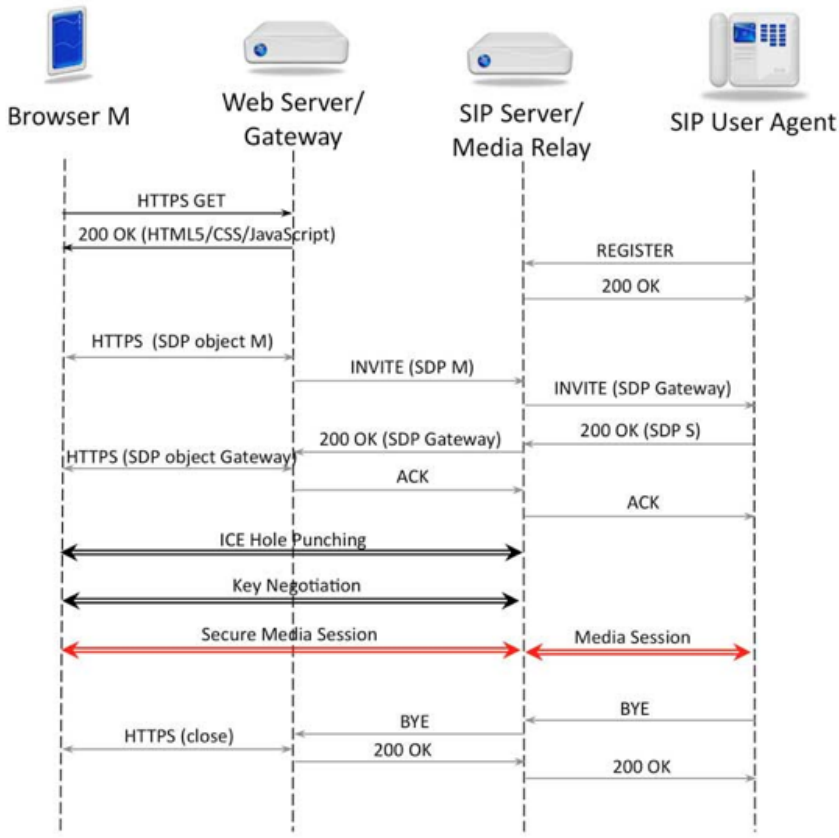

Figure 1.5: Prototype System Working Diagram [JB13c]

Fuel company, which is called PowerMedia XMS v2.1[8] PowerMedia XMS acts as a WebRTC Media Gateway to mediate WebRTC media-plane differences from those of typical existing VoIP networks including encryption interworking, transcoding, and client-based NAT traversal support. The reason to use this media server is to avoid hard-code transition between WebRTC SDP and SIP SDP. Then the end client no matter it is WebRTC client or SIP client, they will communicate with the same signaling client for their aspect.

Moreover, since the media server is used in this case, during the multiple end-point conversation, each end-point will only exchange their media stream to the single end-point on the media server (PowerMedia XMS server), it will make light client

[8]PowerMedia XMS is pre-integrated with a variety of application servers and signaling gateways with HTTP-to-SIP (H2S) functionality and rapidly integrates with others using its web API or standard interfaces.

and centralized media server control. The benefit of this system architecture will be discussed more in the Chapter 2.

Therefore, in the Figure 1.5, all the end point keep using their own original signaling protocol to communicate with different server in order to reach different scope end point.

Chapter 2
System Development

In this Chapter, it will cover research about current WebRTC usage on real-time communication scenario and development progress of the prototype system along with explanation and analysis.

2.1 WebRTC Current Usage

In May 2011, Google released an open source project for browser-based real-time communication known as WebRTC. This has been followed by ongoing work to standardise the relevant protocols in the IETF and browser APIs in the W3C. Then more and more web application are using it in different ways. There are mainly two part of the WebRTC APIs could be used separately or cooperatively in the different web application.

- **MediaStream:** get access to data streams, such as from the user's camera and microphone.
- **RTCPeerConnection:** audio or video calling, with facilities for encryption and bandwidth management.
- **RTCDataChannel:** peer-to-peer communication of generic data.

Because most of the application need to get the user's camera view and microphone sound, the *MediaStream* API is used always in real-time communication application. Normally *MediaStream* API will be used along with *RTCPeerConnection* for showing remote peer media source content. The following business usage cases, 'Tropo' and 'Uberconference', are in this category.

2.1.1 Tropo

Tropo is an application platform that enables web developers to write communication applications in the languages they already use, Groovy[1], Ruby[2], PHP: Hypertext Preprocessor (PHP)[3], Python[4] and JavaScript[5], or use a Web API which will talk with an application running on your own server through the use of HTTP and JavaScript Object Notation (JSON), feeding requests and processing responses back and forth as needed. Tropo is in the cloud, so it manages the headaches of dealing with infrastructure and keeping applications up and running at enterprise-grade. With Tropo, developers can build and deploy voice and telephony applications, or add voice to existing applications.[Cru14a]

It has some advanced features, like 'Phone numbers around the world', 'Text messaging', 'Transcription', 'Call Recording', 'Conferencing', 'Text to Speech' and 'Speech Recognition'. The prototype system in this thesis will provide similar functions like 'Text messaging' and 'Conferencing'. Since Tropo is a cloud application platform, it generates its own scripts based on programming language to provide developer possibility to easily use WebRTC to communicate with other kinds of network rather than IP network. The functions Tropo provided is implemented in application server in the prototype, the application server will handle both the SIP stack and WebRTC stack in the system. For the client scripts will be host on the same application server for browser user to access and use.

2.1.2 Uberconference

UberConference fixes all the broken and outdated aspects of traditional conference calling, making it a more productive business tool, and transforming an industry that hasn't seen real innovation in decades. UberConference gives a visual interface to every conference call so callers can know who's on a call and who's speaking at any time, in addition to making many other features, such as Hangouts[6] integration and screen sharing, easy-to-use with the click of a button. Built by the teams that brought Google Voice[7] and Yahoo! Voice to tens of millions of users, UberConference launched in 2012 and is funded by Andreessen Horowitz and Google Ventures.[Cru14b]

[1]Groovy is an object-oriented programming language for the Java platform. It is a dynamic language with features similar to those of Python, Ruby, Perl, and Smalltalk.[Wik14f]

[2]Ruby is a dynamic, reflective, object-oriented, general-purpose programming language. It was designed and developed in the mid-1990s by Yukihiro "Matz" Matsumoto in Japan.[Wik14n]

[3]PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language.[Wik14k]

[4]Python is a widely used general-purpose, high-level programming language.[Wik14m]

[5]JavaScript (JS) is a dynamic computer programming language.[Wik14i]

[6]Google Hangouts is an instant messaging and video chat platform developed by Google, which launched on May 15, 2013 during the keynote of its I/O development conference.[Wik14d]

[7]Google Voice (formerly GrandCentral) is a telecommunications service by Google launched on March 11, 2009.[Wik14e]

The prototype system in this thesis is ideally to provide same rich media communication platform as the service provided by UberConference. In February of 2014, UberConference release the new feature which allow user to call into a Google Hangouts session with their mobile phone. The feature is shown in Figure 2.1, Once you have installed the UberConference app in Hangouts, people can join your call via phone with the help of a dedicated number.

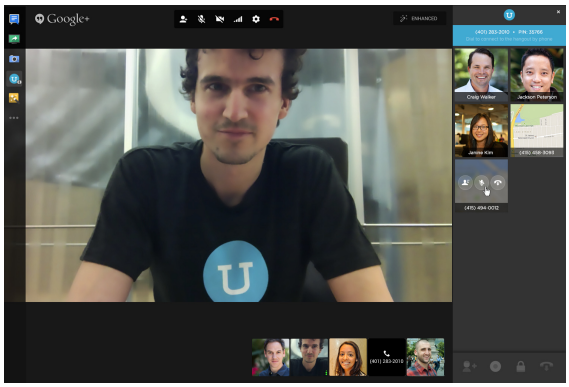

Figure 2.1: UberConference integrate with Hangouts Screen shot[Web14a]

The prototype system will provide the same real-time communication service, but allow the user to create a video conference based on WebRTC on browser by their mobile phone number and communicate with audio only mobile phone user as well. It will be more easier for user since they just need to remember their user credential related to their mobile phone number in order to use the prototype application rather than register another service user binding with private telephone number. During the real-time conversation, the prototype application will provide user cooperation tools like instance message and file sharing in this development phase.

2.1.3 Cube Slam

However, there is another important API, *RTCDataChannel* , can be used more creatively by the developer to build web applications. The experiment usage cases, 'Cube Slam' and 'Webtorrent', are in this category which is using *RTCDataChannel* to build P2P data sharing without data going though the server to dispatch to other peers. It works more efficiently to handle the synchronization problem.

Cube Slam (shown in Figure 2.2) is a Chrome Experiment built with WebRTC , play an old-school arcade game with your friends without downloading and installing any plug-ins. Cube Slam uses *getUserMedia* to access user's webcam and microphone ,*RTCPeerConnection* to stream user video to another user, and *RTCDataChannel* to transfer the bits that keep the gameplay in sync. If two users are behind firewalls, *RTCPeerConnection* uses a TURN relay server (hosted on Google Compute Engine) to make the connection. However, when there are no firewalls in the way, the entire game happens directly peer-to-peer, reducing latency for players and server costs for developers.[Blo14]

Figure 2.2: Cube Slam Game Over Screen

The idea behind the Cube Slam is that use *RTCDataChannel* to sync the player data in real-time to reduce the latency by peer to peer. *RTCDataChannel* sends data securely, and supports an "unreliable" mode for cases where you want high performance but don't care about every single packet making it across the network. In cases like games where low delay often matters more than perfect delivery, this ensures that a single stray packet doesn't slow down the whole app. The prototype application in this thesis will still use WebSocket for data sharing instead of *RTCDataChannel* because the media server using in this system is not support *RTCDataChannel* yet, so it is not possible to create peer to peer session regarding to this issue. This case about using *RTCDataChannel* in prototype application will be discussed in Chapter 4.

2.1.4 Webtorrent

The goal of project Webtorrent is to build a browser BitTorrent client that requires no install, no plugin, no extension, and fully-interoperates with the regular BitTorrent network. It uses WebRTC Data Channels for peer-to-peer transport. Since WebTorrent is web-first, it's simple for users who do not understand .torrent files, magnet links, NATs, etc. By making BitTorrent easier, it will be accessible to new swathes of users who were previously intimidated, confused, or unwilling to install a program on their machine to participate.[Abo14]

Since WebRTC is usually used for peer to peer communication, the *RTCDataChannel* can be used in more creative way like Webtorrent. Although it need to keep the browser up and running on both ends, then there will be no asynchronous

nature into it, it does reduce the bandwidth required and it adds privacy as to who has access to the file being shared. Since the application can reach direct between browsers, it can use the data channel to create a low latency network, where data is shared directly without going through servers on the way. It is lower cost for the developer and more secure on this case. For example, doing the same using a drastically larger number of web browser nodes as The Onion Router (TOR)[8], increases the chance of privacy.This can reduce the need for "real" web servers to run services, and use those only as points of access into the dynamic network that is created ad-hoc.

2.2 WebRTC APIs Implementation

WebRTC components are accessed with JavaScript APIs. Currently in development are the Network Stream API, which represents an audio or video data stream, and the PeerConnection API, which allows two or more users to communicate browser-to-browser. Also under development is a DataChannel API that enables communication of other types of data for real-time gaming, text chat, file transfer, and so forth. Because the media server used in prototype system is not support for DataChannel yet, the DataChannel API will not be covered in this section.

2.2.1 MediaStream API

The MediaStream API represents synchronized streams of media. For example, a stream taken from camera and microphone input has synchronized video and audio tracks. In order to obtain local media, the start step for both peers in Figure 2.3 which is a communication process to set up call process from caller peer, the WebRTC APIs provide *navigator.getUserMedia()* function to get the video and audio stream from user. For privacy reasons, a web application's request for access to a user's microphone or camera will only be granted after the browser has obtained permission from the user. Each MediaStream has an input, which might be a MediaStream generated by *navigator.getUserMedia()*, and an output, which might be passed to a video element or an *RTCPeerConnection*.

The *getUserMedia()* method takes three parameters:

- A constraints object.
- A success callback which, if called, is passed a MediaStream.
- A failure callback which, if called, is passed an error object.

[8]Tor (previously an acronym for The Onion Router) is free software for enabling online anonymity and censorship resistance. Tor directs Internet traffic through a free, worldwide, volunteer network consisting of more than five thousand relays to conceal a user's location or usage from anyone conducting network surveillance or traffic analysis.[Wik14r]

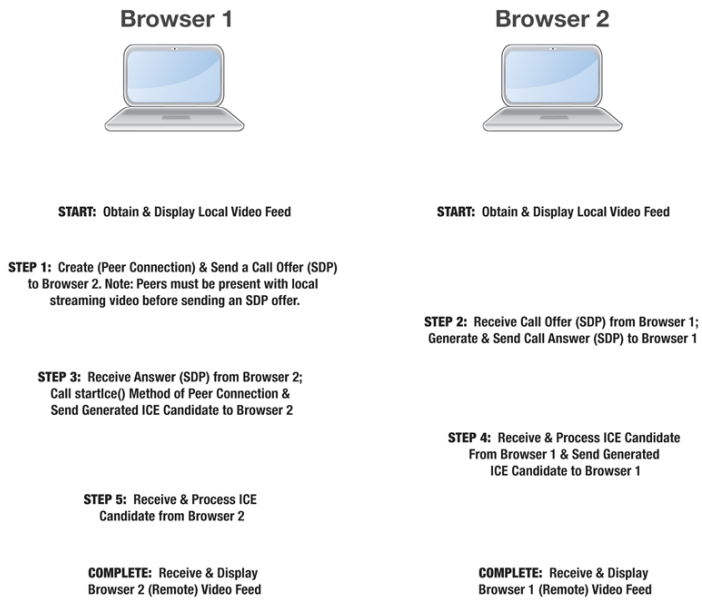

Figure 2.3: WebRTC two peer communication process[Net14]

The Code Snippet 2.1 shows that how the prototype application implements *getUserMedia()* function, it is encapsulated in *WebRTCService* (service is a reusable business logic independent of views in prototype application regarding to AngularJs framework[9]). There will be more discuss about prototype application framework in the later part of this chapter. For the constraints object in parameters, the prototype application set 'audio' and 'video' value to true because it is necessary for the real-time communication application to have video and audio stream both.

```
1 var media_constraints = {audio: true,video: true};
2
3 function _setMediaStream(){
4   WebRTCService.getUserMedia(media_constraints,
5                   _handleUserMedia,
6                   _handleUserMediaError);
7     console.log('Getting user media with constraints',
8           media_constraints);
```

[9]AngularJS is an open-source web application framework, maintained by Google and community, that assists with creating single-page applications, one-page web applications that only require HTML, CSS, and JavaScript on the client side.[Wik14a]

```
9 }
```

Code Snippet 2.1: Get User Media Stream function

getUserMedia() function is currently available in Chrome, Opera and Firefox. Almost all of the WebRTC APIs are slightly different based on different browsers implementation. In the Code Snippet A.1, from line 40 to line 103 is to make all the set up process for FireFox and from line 106 to line 175 is to make the same set up process for Google Chrome. Because WebRTC is not standard Web API yet, so the implementation on different browsers are different and the WebRTC APIs names are slightly different in some browsers. For example, in the Code Snippet A.1 showing, the *RTCPeerConnection* API in Firefox is *mozRTCPeerConnection* but in Google Chrome it is *webkitRTCPeerConnection*. In order to make the WebRTC application works on more browsers, the client side need to figure out which kind of browser is using on the machine then call the corresponding WebRTC APIs. Google provides a JavaScript shim called *adapter.js*. It is maintained by Google, it abstracts away browser differences and spec changes. For Angularjs framework used by prototype application, then the *WebRTCService* is implemented to be integrated with *adapter.js* function to achieve the goal of compatibility.

However, the prototype application in this thesis will only focus on Google Chrome browser[10] to simplify the development process because WebRTC lower level implementation on different browser s are different and hard to track the issues. Then most of the results in this thesis is based on the application performance of Google Chrome browser. The reason to choose Google Chrome browser rather than other browser because WebRTC is the technology rapidly pushed by Google and Google Chrome browser has the most market share in the world. As of March 2014, StatCounter estimates that Google Chrome has a 43% worldwide usage share of web browsers, making it the most widely used web browser in the world.[Wik14c] However, Google changes a lot to improve the performance of WebRTC on Google Chrome browser, then it makes the WebRTC APIs work different on different version of Google Chrome browser. In the Code Snippet A.1, from line 124 to line line 136 is the sample case to distinguish the difference among different version of Google Chrome to handle the *RTCPeerConnection* ICE server constraint implementation.

Since WebRTC APIs is not standard API yet, the prototype application in this thesis will not pay too much work-load on compatibility for different browsers platform. More detail about this issue will be discussed in the Chapter 4.

[10]Google Chrome is a freeware web browser developed by Google. It used the WebKit layout engine until version 27 and, with the exception of its iOS releases, from version 28 and beyond uses the WebKit fork Blink.[Wik14c]

2.2.2 RTCPeerConnection API

To set up peer connection, the *RTCPeerConnection* API sets up a connection between two peers. In this context, "peers" means two communication endpoints on the World Wide Web. Instead of requiring communication through a server, the communication is direct between the two entities. In the specific case of WebRTC, a peer connection is a direct media connection between two web browsers. This is particularly relevant when a multi-way communication such as a conference call is set up among three or more browsers. Each pair of browsers will require a single peer connection to join them, allowing for audio and video media to flow directly between the two peers.

To establish peer connection, it requires a new *RTCPeerConnection* object. The only input to the *RTCPeerConnection* constructor method is a configuration object containing the information that ICE, will use to "punch holes" through intervening NAT devices and firewalls. The Code Snippet 2.2 shows the create *RTCPeerConnection* object and set three listener (*onicecandidate*,*onaddstream*,*onremovestream*) to trigger the handlers to deal with the ICE candidate event and remote stream add/remove events.

The *RTCPeerConnection* API has two arguments to set, one is configuration object for peer connection and the other is constraint object (set transparent protocol and encryption) for peer connection, these value are shown in Code Snippet 2.2 line 1 to line 10. In the showing case, the prototype is using STUN servers for different browser aspect, and set the RTC channel encryption protocol to Datagram Transport Layer Security (DTLS)[11] and enable the RTC DataChannel.

Because in Firefox, WebRTC media transparent channel is only based on DTLS protocol, and in latest version Google Chrome, it is support, then in the prototype application, it will use DTLS protocol to exchange the media stream.

There are two APIs to handle the *IceCandidate* object which contains ICE information data. One is *onicecandidate* listener to trigger the function to handle the new *IceCandidate* data object. The other one is *addIceCandidate* function, which is shown in the Code Snippet 2.3, to add the new *IceCandidate* data object to the remote/local peer connection session description field.

```
1 pc_config = WebRTCService.webrtcDetectedBrowser() === 'firefox' ?
2         {'iceServers':[{'urls':'stun:stun.services.mozilla.com'}]} :
3         {'iceServers':[{'urls': 'stun:stun.l.google.com:19302'}]};
4
5 pc_constraints = {
6         'optional': [
```

[11] In information technology, the Datagram Transport Layer Security (DTLS) protocol provides communications privacy for datagram protocols. DTLS allows datagram-based applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.[Wik14b]

```
7           {'DtlsSrtpKeyAgreement': true},
8           {'RtpDataChannels': true}
9         ]
10      };
11
12  function _createPeerConnection(){
13
14    try {
15      pc = WebRTCService.peerConnection(pc_config, pc_constraints);
16      pc.onicecandidate = _handleIceCandidate;
17      console.log('Created RTCPeerConnnection with:\n' +
18            '  config: \'' + JSON.stringify(pc_config) + '\';\n' +
19            '  constraints: \'' + JSON.stringify(pc_constraints) + '\'.')
                ;
20    } catch (e) {
21      console.log('Failed to create PeerConnection, exception: ' + e.
            message);
22      alert('Cannot create RTCPeerConnection object.');
23      return;
24    }
25    pc.onaddstream = _handleRemoteStreamAdded;
26    pc.onremovestream = _handleRemoteStreamRemoved;
27
28  }
```

Code Snippet 2.2: Create Peer Connection function

```
1  var candidate = WebRTCService.RTCIceCandidate({
2                    sdpMLineIndex:data.content.label,
3                    sdpMid:data.content.id,
4                  candidate:data.content.candidate
5          });
6  pc.addIceCandidate(candidate);
```

Code Snippet 2.3: Add Remote IceCandidate function

In the step 2 of Figure 2.3, after the caller *RTCPeerConnection* run *createOffer()* function to send offer to callee through signaling channel, the callee need run *createAnswer()* function to ask the STUN/TURN server to find the path for each other peer and create the answer with SDP content. SDP is intended for describing multimedia communication sessions for the purposes of session announcement, session invitation, and parameter negotiation. SDP does not deliver media itself but is used for negotiation between end points of media type, format, and all associated properties.[Wik14o] Before *RTCPeerConnection* use *createOffer()* function to send a WebRTC offer to the callee, it is required to be present with local streaming video, like Figure 2.3 mentioned.

The sample SDP from the prototype application is shown in Code Snippet 2.4. Line 2 in Code Snippet 2.4 is the field 'o', it describes originator, session identifier,

username, id, version number and network address. It usually means that where this package comes from. Line 7 and line 17 are field 'm', it describes media name and transport address. And line 11,12 and line 27,28 are the relevant lines for audio and video media field, they describes media filed 'candidate' attributes, in the sample case of Code Snippet 2.4, they are the ICE candidate from the STUN/TURN server. These are important fields regarding to the prototype system because they are used in XMS server and application server of the prototype system.

```
1  sdp: v=0
2  o=xmserver 1399363527 1399363528 IN IP4 10.254.9.135
3  s=xmserver
4  c=IN IP4 10.254.9.135
5  t=0 0
6  a=ice-lite
7  m=audio 49152 RTP/SAVPF 0 126
8  a=rtpmap:0 PCMU/8000
9  a=sendrecv
10 a=rtcp:49153
11 a=candidate:1 1 UDP 2130706431 10.254.9.135 49152 typ host
12 a=candidate:1 2 UDP 2130706430 10.254.9.135 49153 typ host
13 ...
14 a=acfg:1 t=1
15 a=rtpmap:126 telephone-event/8000
16 a=fmtp:126 0-15
17 m=video 57344 RTP/SAVPF 100
18 b=AS:1000
19 a=rtpmap:100 VP8/90000
20 a=fmtp:100 max-fr=30; max-fs=1200
21 a=sendrecv
22 a=rtcp:57345
23 a=rtcp-fb:100 ccm fir
24 a=rtcp-fb:100 nack
25 a=rtcp-fb:100 nack pli
26 a=rtcp-fb:100 goog-remb
27 a=candidate:2 1 UDP 2130706431 10.254.9.135 57344 typ host
28 a=candidate:2 2 UDP 2130706430 10.254.9.135 57345 typ host
29 ...
```

Code Snippet 2.4: Sample WebRTC Answer SDP

In the step 3 of Figure 2.3, the caller will receive the answer from callee and process it by adding the remote SDP to *RTCPeerConnection*, like the Code Snippet 2.3. By the meantime, the step 4 of Figure 2.3, the callee will receive the SDP from caller with the ICE candidate information data, and process it the same way as caller does, add some to *RTCPeerConnection* object by *addIceCandidate()* function.

WebRTC clients (known as peers) also need to ascertain and exchange local and remote audio and video media information, such as resolution and codec capabilities. Signaling to exchange media configuration information proceeds by exchanging an offer

and an answer using the SDP. The *createOffer()* function and *createAnswer()* function both have callback function to handle the SDP either to call *setLocalDescription()* by caller or call *setRemoteDescription()* by callee when callee gets the caller's SDP from WebRTC offer. The Code Snippet2.4 shown is the WebRTC answer SDP from the callee when the callee end-point decide to accept this conversion session.

Once the *RTCPeerConnection* is established, the client need configure where the media or data to store and display if it is necessary. In the prototype application of this thesis, media stream will be displayed in a HTML5 tag called *<video>*. It will only be shown when there is media stream in *<video>* tag source.

2.3 Prototype Implementation Framework

Since WebRTC is a web API, the prototype application will be a web application. There are many different web application framework nowadays to provide rich-client web application. In this section, some of the web application framework will be discussed to figure out which framework is best solution to the prototype scenario. Furthermore, application server is important in the prototype scenario since it does signaling and bridge the SIP network and clients.

2.3.1 Client Implementation Framework

To choose web application framework to implement the client application in this thesis scenario, the main fact is that if the web application framework is fit to the real time communication application and if the framework has the ability to integrate with WebRTC API. After research about these kind of web application framework, it narrows down to three main framework to discuss.

AngularDart :

Sipml5 :

AngularJs + Socket.io:

Chapter 3
System Deployment

Chapter 4
Future Work

4.1 RTCDataChannel usage

4.2 Browser Compatibility

4.3 Media Server Performance

4.4 Object RTC (ORTC) API for WebRTC

4.5 Advanced function for telecommunication

References

[Abo14] Feross Aboukhadijeh. Webtorrent - streaming torrent client for node and the browser, 2014. [Online; accessed 9-May-2014].

[Blo14] The Chromium Blog. Play cube slam, a real-time webrtc video game, 2014. [Online; accessed 8-May-2014].

[Cru14a] Crunchbase. Tropo, 2014. [Online; accessed 8-May-2014].

[Cru14b] Crunchbase. Uberconference, 2014. [Online; accessed 8-May-2014].

[Dut14] Sam Dutton. Getting started with webrtc — html5rocks, 2014. [Online; accessed 2-May-2014].

[Goo12] Google. General overview — webrtc.org, 2012. [Online; accessed 7-May-2014].

[Inc05] Cisco Systems Inc. Differences between traditional telephony and voip, 2005. [Online; accessed 2-May-2014].

[JB13a] Alan B Johnston and Daniel C Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, chapter Preface, page 12. Digital Codex LLC, second edition, 2013.

[JB13b] Alan B Johnston and Daniel C Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, chapter How to Use WebRTC, page 33. Digital Codex LLC, second edition, 2013.

[JB13c] Alan B Johnston and Daniel C Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, chapter How to Use WebRTC, page 48. Digital Codex LLC, second edition, 2013.

[JB13d] Alan B Johnston and Daniel C Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, chapter Introduction to Web Real-Time Communications, page 15. Digital Codex LLC, second edition, 2013.

[Net14] Mozilla Developer Network. Peer-to-peer communications with webrtc, 2014. [Online; accessed 12-May-2014].

[Soc02] The Internet Society. Sip: Session initiation protocol — rfc 3261, 2002. [Online; accessed 7-May-2014].

[Web14a] The Next Web. Uberconference turns google hangouts into a conference calling system, 2014. [Online; accessed 8-May-2014].

[Web14b] Webopedia. Pbx - private branch exchange — webopedia, 2014. [Online; accessed 2-May-2014].

[Wik14a] Wikipedia. Angularjs — Wikipedia, the free encyclopedia, 2014. [Online; accessed 12-May-2014].

[Wik14b] Wikipedia. Datagram transport layer security — Wikipedia, the free encyclopedia, 2014. [Online; accessed 13-May-2014].

[Wik14c] Wikipedia. Google chrome — Wikipedia, the free encyclopedia, 2014. [Online; accessed 12-May-2014].

[Wik14d] Wikipedia. Google hangouts — Wikipedia, the free encyclopedia, 2014. [Online; accessed 8-May-2014].

[Wik14e] Wikipedia. Google voice — Wikipedia, the free encyclopedia, 2014. [Online; accessed 8-May-2014].

[Wik14f] Wikipedia. Groovy (programming language) — Wikipedia, the free encyclopedia, 2014. [Online; accessed 8-May-2014].

[Wik14g] Wikipedia. H.323 — Wikipedia, the free encyclopedia, 2014. [Online; accessed 7-May-2014].

[Wik14h] Wikipedia. Interactive connectivity establishment — Wikipedia, the free encyclopedia, 2014. [Online; accessed 2-May-2014].

[Wik14i] Wikipedia. Javascript — Wikipedia, the free encyclopedia, 2014. [Online; accessed 8-May-2014].

[Wik14j] Wikipedia. List of sip response codes — Wikipedia, the free encyclopedia, 2014. [Online; accessed 7-May-2014].

[Wik14k] Wikipedia. Php — Wikipedia, the free encyclopedia, 2014. [Online; accessed 8-May-2014].

[Wik14l] Wikipedia. Public switched telephone network — Wikipedia, the free encyclopedia, 2014. [Online; accessed 2-May-2014].

[Wik14m] Wikipedia. Python (programming language) — Wikipedia, the free encyclopedia, 2014. [Online; accessed 8-May-2014].

[Wik14n] Wikipedia. Ruby (programming language) — Wikipedia, the free encyclopedia, 2014. [Online; accessed 8-May-2014].

[Wik14o] Wikipedia. Session description protocol — Wikipedia, the free encyclopedia, 2014. [Online; accessed 13-May-2014].

[Wik14p] Wikipedia. Session initiation protocol — Wikipedia, the free encyclopedia, 2014. [Online; accessed 7-May-2014].

[Wik14q] Wikipedia. Skype — Wikipedia, the free encyclopedia, 2014. [Online; accessed 2-May-2014].

[Wik14r] Wikipedia. Tor (anonymity network) — Wikipedia, the free encyclopedia, 2014. [Online; accessed 12-May-2014].

[Wik14s] Wikipedia. Webrtc — Wikipedia, the free encyclopedia, 2014. [Online; accessed 29-April-2014].

[Wik14t] Wikipedia. Websocket — Wikipedia, the free encyclopedia, 2014. [Online; accessed 7-May-2014].

[Wor04] Network World. What is sip? — network world, 2004. [Online; accessed 7-May-2014].

Appendix

Appendix A

A.1 WebRTCService

```
'use strict';

/**
*  services Module
*
* WebRTCService with browser adapter.js function
*/

angular.module('webrtcDemo.services').
  factory('WebRTCService',function () {
    var _ws;//websocket obj

    var _RTCPeerConnection;
    var _RTCSessionDescription;
    var _RTCIceCandidate;
    var _getUserMedia;
    var _createIceServer;
    var _attachMediaStream;
    var _reattachMediaStream;
    var _webrtcDetectedBrowser;
    var _webrtcDetectedVersion;

    function _initWebRTC () {
      _RTCPeerConnection = null;
      _RTCSessionDescription = null;
      _RTCIceCandidate = null;
      _getUserMedia = null;
      _createIceServer = null;
      _attachMediaStream = null;
      _reattachMediaStream = null;
      _webrtcDetectedBrowser = null;
      _webrtcDetectedVersion = null;

      _setRTCElement();
    }
```

```
36
37      function _setRTCElement() {
38
39        if(navigator.mozGetUserMedia){
40          console.log("This appears to be Firefox");
41
42          _webrtcDetectedBrowser = "firefox";
43          _webrtcDetectedVersion = parseInt(navigator.userAgent.match(/
                Firefox\/([0-9]+)\./)[1], 10);
44
45          _RTCPeerConnection = mozRTCPeerConnection;
46          _RTCSessionDescription = mozRTCSessionDescription;
47          _RTCIceCandidate = mozRTCIceCandidate;
48          _getUserMedia = navigator.mozGetUserMedia.bind(navigator);
49
50          // Creates iceServer from the url for FF.
51          _createIceServer = function(url, username, password) {
52            var iceServer = null;
53            var url_parts = url.split(':');
54            if (url_parts[0].indexOf('stun') === 0) {
55              // Create iceServer with stun url.
56              iceServer = { 'url': url };
57            } else if (url_parts[0].indexOf('turn') === 0) {
58              if (_webrtcDetectedVersion < 27) {
59                // Create iceServer with turn url.
60                // Ignore the transport parameter from TURN url for FF
                      version <=27.
61                var turn_url_parts = url.split("?");
62                // Return null for createIceServer if transport=tcp.
63                if (turn_url_parts.length === 1 ||
64                    turn_url_parts[1].indexOf('transport=udp') === 0) {
65                  iceServer = { 'url': turn_url_parts[0],
66                                'credential': password,
67                                'username': username };
68                }
69              } else {
70                // FF 27 and above supports transport parameters in TURN
                      url,
71                // So passing in the full url to create iceServer.
72                iceServer = { 'url': url,
73                              'credential': password,
74                              'username': username };
75              }
76            }
77            return iceServer;
78          };
79
80          _attachMediaStream = function(element, stream) {
81            console.log("Attaching media stream");
82            element.mozSrcObject = stream;
83            element.play();
84          };
```

```
85
86          _reattachMediaStream = function(to, from) {
87            console.log("Reattaching media stream");
88            to.mozSrcObject = from.mozSrcObject;
89            to.play();
90          };
91
92          // Fake get{Video,Audio}Tracks
93          if (!MediaStream.prototype.getVideoTracks) {
94            MediaStream.prototype.getVideoTracks = function() {
95              return [];
96            };
97          }
98
99          if (!MediaStream.prototype.getAudioTracks) {
100           MediaStream.prototype.getAudioTracks = function() {
101             return [];
102           };
103         }
104
105       }else if(navigator.webkitGetUserMedia){
106         console.log("This appears to be Chrome");
107
108         _webrtcDetectedBrowser = "chrome";
109         _webrtcDetectedVersion = parseInt(navigator.userAgent.match(/
              Chrom(e|ium)\/([0-9]+)\./)[2], 10);
110
111         _RTCPeerConnection = webkitRTCPeerConnection;
112         _RTCSessionDescription = RTCSessionDescription;
113         _RTCIceCandidate = RTCIceCandidate;
114         _getUserMedia = navigator.webkitGetUserMedia.bind(navigator);
115
116         // Creates iceServer from the url for Chrome.
117         _createIceServer = function(url, username, password) {
118           var iceServer = null;
119           var url_parts = url.split(':');
120           if (url_parts[0].indexOf('stun') === 0) {
121             // Create iceServer with stun url.
122             iceServer = { 'url': url };
123           } else if (url_parts[0].indexOf('turn') === 0) {
124             if (_webrtcDetectedVersion < 28) {
125               // For pre-M28 chrome versions use old TURN format.
126               var url_turn_parts = url.split("turn:");
127               iceServer = { 'url': 'turn:' + username + '@' +
                      url_turn_parts[1],
128                             'credential': password };
129             } else {
130               // For Chrome M28 & above use new TURN format.
131               iceServer = { 'url': url,
132                             'credential': password,
133                             'username': username };
134             }
```

```
135           }
136           return iceServer;
137         };
138
139         // Attach a media stream to an element.
140         _attachMediaStream = function(element, stream) {
141           if (typeof element.srcObject !== 'undefined') {
142             element.srcObject = stream;
143           } else if (typeof element.mozSrcObject !== 'undefined') {
144             element.mozSrcObject = stream;
145           } else if (typeof element.src !== 'undefined') {
146             element.src = URL.createObjectURL(stream);
147           } else {
148             console.log('Error attaching stream to element.');
149           }
150         };
151
152         _reattachMediaStream = function(to, from) {
153           to.src = from.src;
154         };
155
156         // The representation of tracks in a stream is changed in M26
157         // Unify them for earlier Chrome versions in the coexisting
               period
158         if (!webkitMediaStream.prototype.getVideoTracks) {
159           webkitMediaStream.prototype.getVideoTracks = function() {
160             return this.videoTracks;
161           };
162           webkitMediaStream.prototype.getAudioTracks = function() {
163             return this.audioTracks;
164           };
165         }
166
167         // New syntax of getXXXStreams method in M26
168         if (!webkitRTCPeerConnection.prototype.getLocalStreams) {
169           webkitRTCPeerConnection.prototype.getLocalStreams = function
                 () {
170             return this.localStreams;
171           };
172           webkitRTCPeerConnection.prototype.getRemoteStreams = function
                 () {
173             return this.remoteStreams;
174           };
175         }
176
177       }else{
178         console.log("Browser does not appear to be WebRTC-capable");
179       }
180
181     }
182
183     return {
```

```
184       init : function (socket) {
185         if(socket){
186           //init service with websocket
187           _ws = socket;
188         }
189
190         _initWebRTC();
191
192       },
193
194       peerConnection : function(config,constraints){
195         if(_RTCPeerConnection){
196           return new _RTCPeerConnection(config,constraints);
197         }
198         return null;
199       },
200
201       RTCSessionDescription : function(message){
202         if(_RTCSessionDescription){
203           return new _RTCSessionDescription(message);
204         }
205         return null;
206       },
207
208       RTCIceCandidate : function(options){
209         if(_RTCIceCandidate){
210             return new _RTCIceCandidate(options);
211           }
212           return null;
213       },
214
215       webrtcDetectedBrowser : function(){
216         return _webrtcDetectedBrowser;
217       },
218
219       attachMediaStream : function(element, stream){
220         return _attachMediaStream(element, stream);
221       },
222
223       reattachMediaStream : function(to, from){
224         return _reattachMediaStream(to, from);
225       },
226
227       getUserMedia : function(constraints, handleUserMedia,
              handleUserMediaError){
228         return _getUserMedia(constraints, handleUserMedia,
              handleUserMediaError);
229       }
230
231     }
232   });
```

Code Snippet A.1: WebRTCService in application client