Adding Real-time Video Conferencing to QT Web kit Browser

DhavalKumar Sharma[1] Shankar Patel[2]
[1,2]M.E. Student
[1]Department of Computer Engineering
[1]Gujarat Technological University, India [2]IISc Bangalore, Lexis Technology, Gandhinagar, India

Abstract— The project explores the use of WEBRTC module to implement real time Audio/Video communication over web in mobile devices without using any additional applications or plug-in. Most Mobile devices use QT as an application development framework to create various applications. Similarly QT web kit is the most popular web browser for mobile devices for connecting and accessing internet and websites. However, till now it does not have any inherent support for real time audio/video communication support to communicate with people over internet in real time. Hence, people have to use third party separate plug-in or applications such as Facebook or Skype for this purpose. This has lot of drawbacks and the overall efficiency, quality and speed is less. On the other side the world is shifting towards open source software development platform. WEBRTC is one such open source software module which provides real time audio/video capabilities in to the browsers so that the users can communicate with other people on internet without using any third party application or plug-in. The aim of this project is to use this open-source software module and integrate it with QT web kit browser so that mobile device users can communicate with each other in real time without the need of any additional third party application or plug-in. This is much more efficient and speedy and the quality of the real time communication is also much better.

Key words: Point to Point Audio/Video Conferencing, Communication/QT Application Framework for Mobile Application Development/ Embedded Linux Operating System/WEBRTC Module for Audio/Video support in browsers over web/porting WEBRTC to QT

I. INTRODUCTION

Until recently, the ability to play any type of audio/video within a browser involves downloads, native apps or plug-ins. These include Skype, Face book (which uses Skype) and Google Hangouts (which use the Google Talk plug-in) or Adobe flash. Although Adobe's Flash player is unquestionably the most ubiquitous of these, most developers and designers would agree that it's better not to rely on plug-ins at all.

Fig.1

Browsers were like person not having their own eyes, ears and mouth. They had to rely on third party software tools which were proprietary in nature. This leads live streaming based html applications difficult, expensive and could not easily be integrated in different browser. Application written for one browser will not work in different browser.

Now a days many low cost embedded devices are coming in market like Raspberry Pi, Beagle Boneblack, Panda Board etc. and there are many applications getting developed around these devices including home security, real time communication and many more which uses camera, microphone, speaker etc. QT web kit browser is one of the best suitable browsers for embedded devices. If audio/video is natively supported in browser, then very efficient, low cost and nice application can be developed using these devices.

So it's best if browser itself has support for audio/video natively.

QT does not only provide browse but it also provide windowing environment on which graphical user interface (GUI) based applications are developed. Before Android become popular recently, QT was ultimate choice for smart phone like devices. QT is also very popular in set top box (STB) devices to implement smart TV application. Most of the big TV vendors now days use QT and web kit browser as GUI application development platform. The channel guide basically we see on our TV, they are HTML applications running on QT browser. [1]

Lexis tech was working on a research project for a TV vendor on Linux/QT based STB devices to develop a solution to help their customers and resolve their issues on real time. Generally when there is any issue happens on customer's TV box, they call customer care center over phone and explain their issue. Most of the users being no-technical, they do not understand technical person's language to troubleshoot the issue and it became frustrating to both user and customer care representative. So it was thought process to develop some relative video application so that when customer talk to customer care representative, he can a video call and share his TV screen to customer care so that they can quickly resolve the issues.

The project is mainly targeted for US market where every home has broadband internet connection and STB is also connected with internet. There are many choices available to develop real-time video platform on STB like Skype, Adobe flash etc but these being proprietary in nature are not optimal choice for STB which is a low cost device and user will not pay extra money just for troubleshooting technical issues. WEBRTC which is open source developed by Google was the best choice.

In this paper we are going to discuss WEBRTC and implementing the support in QT web kit. WEBRTC is a free, open project that enables web browsers with Real-Time Communications (RTC) capabilities via simple JavaScript APIs. It is an effort to build browsers with their own eyes, ear and mouth. This is started by Google, to build a standard based real time Media Engine into all of the available browsers.

Fig.2

WEBRTC offers web application developers the ability to write rich, real-time multimedia applications (think video chat) on the web, without requiring plug-ins, downloads or installs. Its purpose is to help build a strong RTC platform that works across multiple web browsers, across multiple platforms.

With WEBRTC in a browser, a web services application can now instruct the browser to make a real time voice or video connection to another WEBRTC device or to a WEBRTC media server using RTP. With a HTML5 and WEBRTC enabled Browser, a soft client is now just HTML pages from sever as the visual interface with WEBRTC APIs and Media Engine to define the communications path.[2]

- WEBRTC is an open-source application programming interface (API)
- WEBRTC enables any Web server to deliver a unique real-time communications experience, with simplicity and reliability, without dependence on service providers or other services
- WEBRTC enables users to participate in a communications experience as delivered by any website without downloads, registration or general cost
- WEBRTC is Platform and device independence
- Secure voice and video
- Rapid application development

II. LITERATURE REVIEW

A. *Approach and thought process*

Explain initial thought process to how you are trying to address problem and what you aim to achieve at the end of the project.

1) IEEE Paper#1

Yu-Chung Wang and Kwei Jay Lin, "Enhancing real-time capability of Linux-Kernel", fifth international conference on digital object identifier, 1998.

a) Concept

Linux has become one of the most popular UNIX operating systems. Commercial support for Linux and adoption of Linux in real world applications has now started to emerge. Some of the applications have strict real-time requirements. The real-time capability is achieved by three new kernel mechanisms. The porting a micro timer into the kernel is done. The time-driven scheduling paradigm and implement a time-driven scheduler in the Linux kernel. Finally, by inserting preemption points in the kernel to make it more pre-expiable so that real-time jobs may experience shorter blockings. This is general approach and may be used to enhance the real-time capability of other non-real-time operating systems as well.

b) Summary

I study an approach to convert a traditional monolithic kernel into a real-time operating system without extensive re-implementation which is comparable in performance to those fully-pre-emptive real-time kernels. I also studied that in UNIX, the priorities of interrupt handlers are higher than other processes but a real-time system, and interrupts should be scheduled along with other real-time processes according to their priorities. Interrupts should inherit priorities from the processes that generate or wait for them. [3]

2) IEEE Paper #2

M.V. Panduranga Rao, K.C. Shet ,R.Balakrishna ,K. Roopa "Development of Scheduler for Real Time and Embedded System", 22nd International Conference on Advance Information Networking and Applications – Workshops,2008. Page 3

a) Concept

Though there are several scheduling policies, the preemptive scheduling policy holds promising results. In this research paper, the different approaches to design of a scheduler for real-time Linux kernel are discussed in detail. The comparison of different preemptive scheduling algorithms is performed. Hence, by extracting the positive characteristics of each of these preemptive scheduling policies, a new hierarchical scheduling policy is developed. The proposed hierarchical scheduling for real time and embedded system will be implemented for a prototype system, using C or C++ language. It is expected that the new scheduling algorithm will give better performance with respect to satisfy the needs, such as time, capturing and usage of resources of different applications.

b) Summary

I study the policy mechanisms of different real-time schedulers in embedded domain. The main aim is to study different performance of this real time scheduling mechanisms. Study implementation of a good, robust, fully pre-emptive real-time scheduler. [3]

3) IEEE Paper #3

Xu Zhe,Liu Zhou, Zhang Hua, "Development of Linux based USB device driver for portable spectrometer" Control and Decision Conference, 2009.

a) Concept

USB device interface on the PXA270 processor is designed on demand of data transmission for portable spectrometer analyzing gadget system architecture in Linux system, USB device driver is realized and the function of mass storage provided by gadget is used for spectrometer to compliments its function of USB storage device so that spectral data file can be transferred easily from portable spectrometer to personal computer.

b) Summary

I study USB device interface on the PXA270 processor for data transmission from portable spectrometer to PC.USB device driver is realized and the function of Page 4
mass storage provided by gadget is used for spectrometer to compliments its function of USB storage device , so that spectral data file can be transferred easily from portable spectrometer to PC. [3]

4) IEEE Paper #4

YE Dun-fan, ZHOU Fei-fan, MIN Liang-liang "Design and Implementation of High-Precision Timer in Linux", World Congress on Computer Science and Information Engineering, 2009.

a) Concept

Clock precision directly affects that if tasks can be timely responded and scheduled or not. As a time-sharing system, Linux cyclical time granularity cannot achieve microsecond response precision. Simply raising clock frequency means that the clock interrupt's process will take up too much time of processor. And it makes the system's effective utilization decline. This paper introduces some ways of improving clock's precision for real-time Linux, analyses the theory of MontaVista Linux high resolution timer (HRT) mechanism, implements a high-precision clock in current popular embedded operating systems through modifying Linux kernel to design a Linux high-precision timer to raising the Linux clock precision effectively.

b) Summary

I study how Clock precision directly affects that if tasks can be timely responded and scheduled or not. In this paper, I study some ways of improving clock's precision for real-time Linux and study analysis of the theory of Monta Vista Linux high-resolution timer (HRT) mechanism.[3]

5) IEEE Paper #5

Irena Trajkovska, Pedro Rodríguez, Javier Cerviño and Joaquín Salvachúa"OPPORTUNITIES AND CHALLENGES OF IMPLEMENTING P2P STREAMING APPLICATIONS IN THE WEB"Universidad Politécnica de Madrid Ciudad Universitaria, Avda. Complutense s/n 28040, adrid, Spain

a) Concept

P2P applications are increasingly present on the web and identified a gap in current proposals when it comes to the use of traditional P2P overlays for real-time multimedia streaming. also analyze the possibilities and challenges to extend WebRTC in order to implement JavaScript APIs for P2P streaming algorithms.

b) Summary

Analyzed the opportunities and challenges of supporting P2P streaming applications using WebRTC,suggesting an implementation of current P2P algorithms in JavaScript. Possible use case of this outcome could be a web adaptation of various P2P architectures.Further proposal could include JavaScript APIs for QoS management in P2P architectures. However additional changes would be required in the current WebRTC standard to overcome the challenges described throughout,this paper facilitate the creation of more developer- and user-friendly P2P applications on the web.[3]

6) IEEE Paper #6

Alan Johnston,Distinguished Engineer,Avaya, Dan Burnelt,Director of standards, Voxeo labs "WebRTC: The web way to communicate" webRTC IEEE Louis & COMSOC April 2013.

a) Concept

This paper shows the basic guidelines of what the webrtc is and how it works in peer to peer connection also the advantages and standards to use it in browser to browser communication.

b) Summary

It concludes that there is the research work is available on this kind of project and the implementation on mobile device's browser is still being a cake topic for researchers. And also market client(enterprises) still need help from developer to implement such kind of technology with great result of quality and reliability and possibilities of use in enterprises.[3]

B. Feasibility Study of implementing wetrtc in QT browser

Buis& ceing a customer need to implement real time video platform in QT browser, we need to do feasibility study before we can start on implementation part because we do not have option to revert back and have to be time to market. QT just like anny windowing platform is very big and complex. Even compilation of QT takes 4-5 five hours on dual core machine, so we have to be very smart for all the steps we do.

The first feasibility we need to do is whether STB can support USB kind of webcam devices. We need to check both software and hardware to support USB based webcam devices. The generic Linux on running on STB does not support camera devices. We rebuild the kernel with V4L2 kernel subsystem and port USB video class (UVC) driver for USB based webcam devices. We tested many webcam devices successfully recognized by STB and gave enough frame rate for real time video application.

We used standard V4L2 test application (http://v4l-test.sourceforge.net/) to verify the quality and frame rate. The test application captures the raw images from camera, convert in JPG format and write into disk.We verified various resolution (320 x 480, 640 x 480, 1280 x 720) and frame rate (10 fps to 40 fps).

Second important feasibility was to test video streaming capability over network. For that we need to setup streaming server on the STB box. The STB already supports gstreamer which can stream video over network, but this was not having v4l2 plugin required for webcams. Gstreamer is a pipeline base multimedia framework and all the components like encoder, decoder, and multiplexer acts as pipeline. We compiled gstreamer with v4l2 and test the streaming capability. We used vlc player as client.[4]

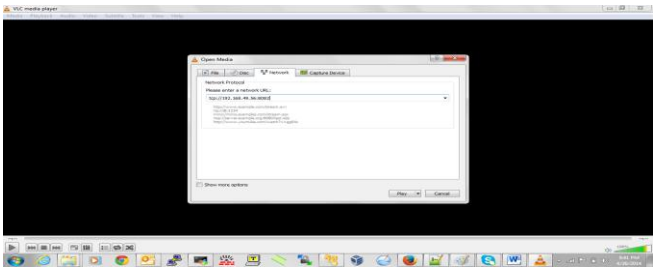

Fig.3

The above two feasibility test helps to verify STB hardware and Linux capability for real time audio/video streaming. Next we need to verify if webrtc can be ported in QT. There are few points we need to check…

- The language of QT and webrtc
- Memory requirement of webrtc
- Audio codec
- Video codec

Both QT and webrtc are written in c/c++ so webrtc can be ported but we need to do effort estimate. It need to understand the both QT and webrtc before we start implementing.

The audio codec "Opus" used in webrtc supports constant and variable bitrate encoding and requires 6–510 Kbit/s of bandwidth. The good news is that the codec can switch seamlessly and adapt to variable bandwidth.

The video codec "VP8" used for webrtc encoding requires 100–2,000+ Kbit/s of bandwidth, and the bitrate depends on the quality of the streams:

- 720p at 30 FPS: 1.0~2.0 Mbps
- 360p at 30 FPS: 0.5~1.0 Mbps
- 180p at 30 FPS: 0.1~0.5 Mbps

All these parameters are very much under h/w STB limit.[4]
Effort estimate for integrating webrtc in QT webkit.

Task	**Detail**	**Estimate (Man-days)**
Feasibility and POC	- V4L2 support in Linux - Gstreamer with V4L2 support - Codec requirement	30
Study QT components	- QT build framework - QT media framework - QT webkit module - QT logging and Debugging	10
Study QT Webrtc	- Chrome/Mozilla source code - Webrtc Web APIs - V4L2 layer - RTP layer - Codec (Opus/VP8)	15
Build Environment setup	- Create VM and install required tools - Compile QT	5
Build webrtc with QT	- Create .pro and .pri files for all components of webrtc - Add optionin QT build framework to compile webrtc as QT module - Fix compilation issues	12
Write web adaptation layer	- Write integration layer to plugin webrtc APIs in QT webkit	30
Write test app	- Test app to verify user media API - Test app to verify Peer to peer API	5

Table.1

III. PROJECT OUTLINE

A. Revisiting Problem statement

QT is a cross-platform application and UI framework for developers using C++ or QML, a CSS & JavaScript like language. QT is ideal for embedded development and provides comprehensive C++ development libraries and tools needed to create amazing user experiences for leading platforms on desktop, embedded and mobile.QT framework also has inbuilt support for WebKitbrowser is an open source web browser engine with.

- State of the art rendering engine
- Css/svg support
- Super-fast js engine
- Plug-in support

Many of the embedded devices like mobile phones, set top box, gaming devices uses QT and GUI platform. It's not very easy to develop rich applications which need audio/video in real time. The applications developed for desktop environment can also not be easily ported. But if we enable audio/video capability in QT webkit itself then the remove the above limitation and users can port their rich applications developed for desktop browsers to embedded devices as well without any extra cost. [5]

B. Aim

In this project we want to show that QT webkit can be enabled with native audio/video capabilities which can be used to develop rich html applications using audio and video.

C. Current Situation

WebRTC, the module which implement audio/video capability natively in browsers has already been ported in many browsers like Google chrome, Mozilla Firefox, and opera. Many people have tried to port WebRTC in QT but with limited success. In this project we will port WebRTC in QT and make QT browser taking to Google chrome and Mozilla Firefox. We will also demonstrate the applications running on Chrome/Mozilla Firefox can directly be ported on QT.

D. WEBRTC ARCHITECTURE

WebRTC offers web application developers the ability to write rich, real-time multimedia applications on the web, without requiring plug-ins, downloads or installs. Its

purpose is to help build a strong RTC platform that works across multiple web browsers, across multiple platforms.[6]

The overall architecture looks something like this:

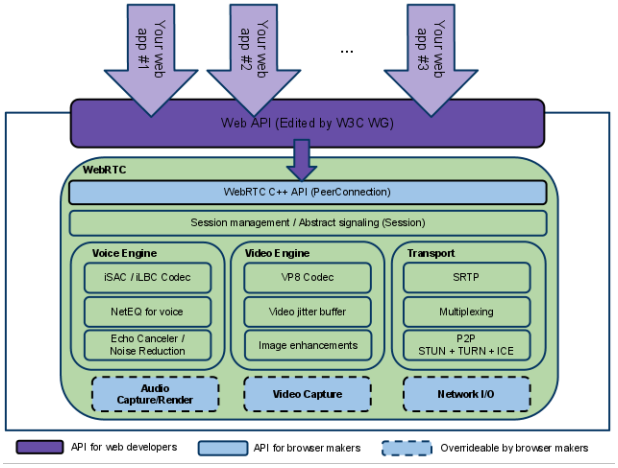

Fig.4

1) Web Application

A third party developer web based application with video and audio chat capabilities powered by the web API for real time communications.

2) Web API

An API to be used by third party developers for developing web based video chat-like applications.

3) WebRTC Native C++ API

An API layer that enables browser makers to easily implement the Web API proposal

4) Transport / Session

The session components are built by re-using components from libjingle, without using or requiring the xmpp/jingle protocol.

5) RTP Stack

A network stack for RTP, the Real Time Protocol

6) STUN/ICE

A component allowing calls to use the STUN and ICE mechanisms to establish connections across various types of networks.

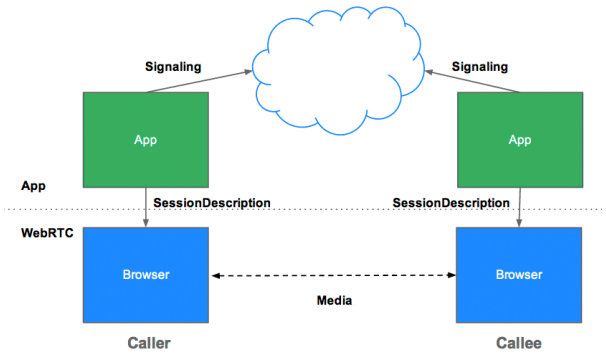

Fig.5

7) Session Management

An abstracted session layer, allowing for call setup and management layer. This leaves the protocol implementation decision to the application developer.

8) VoiceEngine

VoiceEngine is a framework for the audio media chain, from sound card to the network.

9) iSAC / iLBC / Opus

- iSAC: A wideband and super wideband audio codec for VoIP and streaming audio. iSAC uses 16 kHz or 32 kHz sampling frequency with an adaptive and variable bit rate of 12 to 52 kbps.
- iLBC: A narrowband speech codec for VoIP and streaming audio. Uses 8 kHz sampling frequency with a bit rate of 15.2 kbps for 20ms frames and 13.33 kbps for 30ms frames. Defined by IETF RFCs 3951 and 3952.
- Opus: Supports constant and variable bitrate encoding from 6 kbit/s to 510 kbit/s, frame sizes from 2.5 ms to 60 ms, and various sampling rates from 8 kHz (with 4 kHz bandwidth) to 48 kHz (with 20 kHz bandwidth, where the entire hearing range of the human auditory system can be reproduced). Defined by IETF RFC 6176.[10]

10) NetEQ for Voice

A dynamic jitter buffer and error concealment algorithm used for concealing the negative effects of network jitter and packet loss. Keeps latency as low as possible while maintaining the highest voice quality.

11) Acoustic Echo Canceller (AEC)

The Acoustic Echo Canceller is a software based signal processing component that removes, in real time, the acoustic echo resulting from the voice being played out coming into the active microphone.[6]

12) Noise Reduction (NR)

The Noise Reduction component is a software based signal processing component that removes certain types of background noise usually associated with VoIP. (Hiss, fan noise, etc...)[6]

13) Video Engine

Video Engine is a framework video media chain for video, from camera to the network, and from network to the screen. [6]

14) VP8

Video codec from the Web Project. Well suited for RTC as it is designed for low latency.

15) Video Jitter Buffer

Dynamic Jitter Buffer for video. Helps conceal the effects of jitter and packet loss on overall video quality.

16) Image enhancements

For example, removes video noise from the image capture by the webcam.

IV. WEBRTC APPLICATION

- Call a friend
- Video conferencing / sharing slides
- Games that integrate video, audio and data sharing.
- Remote assistance/co-pilot

V. WEBRTC APIS FOR HTMP APP DEVELOPERS

A. Media Stream (getUserMedia)

Get access to data streams, such as from the user's camera and microphone.

getUserMedia is an API that gives a web page access to a user's camera and microphone via JavaScript. The real code is implemented in c++ in web adaptation layer. HTML application writer can write java scrip to call this function.

This function takes three arguments… constraint and two callback functions.The constant parameter is used to specify property of media stream like if both audio/video needed, resolution of video etc.

The first callback is called when stream is successfully created matching the constraints. This is called successful callback.

The second callback function is called when stream was not created successfully may be because of any internal error or constraints do not match. This is called error callback.[7]

API looks like…

navigator.getUserMedia (constraints, successCallback, errorCallback);

Below code shows sample javascrip for getUserMedia API…

```
 function requestVideo() {
  getUserMedia({video: true, audio: false},
                getUserMediaSuccessCallback,
                getUserMediaEroorCallback);
 }
 function getUserMediaErrorCallback(error) {
  alert("User media request denied with error code " + error.code);
 }
 function getUserMediaSucessCallback(stream) {
  // Call the polyfill wrapper to attach the media stream to this element.
  attachMediaStream(document.getElementById("view1"), stream);
 }
```

This API will be called in HTML code. Sample HTML code will look like…

```
<body onload="requestVideo();">
<table border="0">
<tr>
<td>Local Preview</td>
</tr>
<tr>
<td><video width="320" height="240" id="view1"
      autoplay="autoplay"></video></td>
</tr>
</table>
</body>
```

B. RTCPeerConnection (RTCPeerConnection API)

Audio or video calling, with facilities for encryption and bandwidth management.

This API handles the peer-to-peer connection between two browsers. Stream created by getUserMedia API is send as offer to remote browser. When remote browser accepts the request, it create its local stream and send back to calling browser. Calling bowser need to create anRTCPeerConnection object and add local stream to create offer for remote browser.[12]

lpc = new RTCPeerConnection(configuration);

The argument"configuration" includes STUN and TURN server information. There may be multiple servers of each type and any TURN server also acts as a STUN server.

An RTCPeerConnection object has an associated ICE agent [ICE], RTCPeerConnection signaling state, ICE gathering state, and ICE connection state. These are initialized when the object is created.

Next, calling browser need to create offer for remote device using createOffer and send to remote browser.[8]

Below is sample code to create peer to peer connection:

```
Var localStream, localPeerConnection, remotePeerConnection;
var localVideo = document.getElementById("localVideo");
var remoteVideo = document.getElementById("remoteVideo");
localVideo.addEventListener("loadedmetadata", function(){
trace("Local video currentSrc: " + this.currentSrc +
      ", videoWidth: " + this.videoWidth +
      "px,  videoHeight: " + this.videoHeight + "px");
});
remoteVideo.addEventListener("loadedmetadata", function(){
trace("Remote video currentSrc: " + this.currentSrc +
      ", videoWidth: " + this.videoWidth +
      "px,  videoHeight: " + this.videoHeight + "px");
});
var startButton = document.getElementById("startButton");
var callButton = document.getElementById("callButton");
var hangupButton = document.getElementById("hangupButton");
startButton.disabled = false;
callButton.disabled = true;
hangupButton.disabled = true;
startButton.onclick = start;
callButton.onclick = call;
hangupButton.onclick = hangup;
var total = '';
```



```
function trace(text) {
 total += text;
 console.log((performance.now() / 1000).toFixed(3) + ": " + text);
}

function gotStream(stream){
 trace("Received local stream");
 localVideo.src = URL.createObjectURL(stream);
 localStream = stream;
 callButton.disabled = false;
}
function start() {
 trace("Requesting local stream");
 startButton.disabled = true;
 navigator.getUserMedia = navigator.getUserMedia ||
  navigator.webkitGetUserMedia || navigator.mozGetUserMedia;
 navigator.getUserMedia({video:true}, gotStream,
  function(error) {
   trace("navigator.getUserMedia error: ", error);
  });
}
function call() {
 callButton.disabled = true;
 hangupButton.disabled = false;
 trace("Starting call");
 if (localStream.getVideoTracks().length > 0) {
  trace('Using video device: ' + localStream.getVideoTracks()[0].label);
 }
 if (localStream.getAudioTracks().length > 0) {
  trace('Using audio device: ' + localStream.getAudioTracks()[0].label);
 }
var servers = null;
localPeerConnection = new webkitRTCPeerConnection(servers);
trace("Created local peer connection object localPeerConnection");
localPeerConnection.onicecandidate =gotLocalIceCandidate;
remotePeerConnection = new webkitRTCPeerConnection(servers);
trace("Created remote peer connection object remotePeerConnection");
remotePeerConnection.onicecandidate = gotRemoteIceCandidate;
remotePeerConnection.onaddstream = gotRemoteStream;localPeerConnection.addStream(localStream);
trace("Added localStream to localPeerConnection");
 localPeerConnection.createOffer(gotLocalDescription);
}

function gotLocalDescription(description){
localPeerConnection.setLocalDescription(description);
trace("Offer from localPeerConnection: \n" + description.sdp);
remotePeerConnection.setRemoteDescription(description);
remotePeerConnection.createAnswer(gotRemoteDescription);
}
function gotRemoteDescription(description){
 remotePeerConnection.setLocalDescription(description);
 trace("Answer from remotePeerConnection: \n" + description.sdp);
 localPeerConnection.setRemoteDescription(description);
}
function hangup() {
 trace("Ending call");
 localPeerConnection.close();
 remotePeerConnection.close();
 localPeerConnection = null;
 remotePeerConnection = null;
 hangupButton.disabled = true;
 callButton.disabled = false;
}
function gotRemoteStream(event){
 remoteVideo.src = URL.createObjectURL(event.stream);
 trace("Received remote stream");
}

function gotLocalIceCandidate(event){
 if (event.candidate) {
  remotePeerConnection.addIceCandidate(new RTCIceCandidate(event.candidate));
  trace("Local ICE candidate: \n" + event.candidate.candidate);
 }
}
function gotRemoteIceCandidate(event){ if (event.candidate) { localPeerConnection.addIceCandidate(new RTCIceCandidate(event.candidate));
trace("Remote ICE candidate: \n " + event.candidate.candidate);
```


}

}

C. RTCDataChannel

Peer-to-peer communication of generic data.

1) Proposed System design

- Conceptual block diagrams
- Explanation of block diagram

VI. IMPLEMENTATION

Implementation of Real time video conferencing in QT webkit will involve followings.

- Setting up the build environment on Ubuntu host to compile and run QT
- Getting latest QT source code from open source
- Getting latest Googlechrome webrtc code from open source
- Create/Modifying QT project files to add compilation support for Google webrtc as QT module.
- Taking the Google chrome browser source code as reference, adding support for media (audio/video) handling in QT and integrating webrtc APIs in QT.
- Compiling QT with webrtc enabled
- Using Google chrome's webrtc demo to verify on QT browser to make real time video call.

A. Setting up the build environment for QT

This section provides the information to setup build environment (hardware and software) to compile QT from source code. QT being cross platform framework supports compilation on Windows, Linux, Mac and other operating system. There are many flavors in Linux like Ubuntu, Fedora, Opensuse etc. Ubuntu especial 12.04, 32 bit LTS is very stable and user friendly and widely supported by open community and hence we have selected Ubuntu as build environment for QT.[9]

Ubuntu is also very much supported in virtual environment like vm-ware and virtual box. So we can install Ubuntu on using vm-ware on our windows machine and can run Ubuntu as well as Windows together. It will also facilitate development and testing. We can use same computer to make a video call between QT (running on Ubuntu on virtual environment and Google Chrome running on windows.

We will have to use two cameras, one for Chrome (on Windows) and 2nd for QT (on Ubuntu). The best part of vm-ware is that we can be connected many hardware in vm-ware also. Below is the procedure to setup vm-ware and installing Ubuntu from iso.[10]

- Download vm-ware for windows and from below link and install.
  https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/6_0
- Download Ubuntu 12.04 LTS for 32 bid desktop from
  http://releases.ubuntu.com/12.04/
- Follow the instructions at http://www.howtogeek.com/howto/11287/how-to-run-ubuntu-in-windows-7-with-vmware-player/ to install Ubuntu on vm-ware.

Now we have Ubuntu build machine and we can setup the build environment for QT. We have followed environment setup instruction given at official QT website http://qt-project.org/wiki/Building-Qt-5-from-Git specific to Ubuntu. Below are the brief of instructions[11]

- sudo apt-get install build-essential perl python git
- sudo apt-get install "^libxcb.*" libx11-xcb-dev libglu1-mesa-dev libxrender-dev
- sudo apt-get install flex bison gperf libicu-dev libxslt-dev ruby
- sudo apt-get install libasound2-dev libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev
- sudo apt-get install libicu-dev

Now we have proper QT build environment. We will now create workspace where we can download all the source codes and compile…

- mkdir /home/dhaval/qtrtc
- mkdir /home/dhaval/qtrtc/build

B. Getting the QT source code

I have taken the latest stable QT source code (qt-everywhere-opensource-src-5.0.1.tar.gz) from and extract in /home/dhaval/qtrtc

https://download.qt-roject.org/archive/qt/5.0/5.0.1/single/qt-everywhere-opensource-src-5.0.1.tar.gz.mirrorlist

- cd /home/dhaval/qtrtc
- tar zxvf /<path of qt tar>/qt-everywhere-opensource-src-5.0.1.tar.gz -C .

The above command will extract all the QT source code in /home/dhaval/qtrtc/qt-everywhere-opensource-src-5.0.1

C. Getting the WEBRTC source code

We have taken latest source code from Google's git repository …

http://git.chromium.org/gitweb/?p=external/webrtc.git;a=summary

The code is also available at svn repo at http://webrtc.googlecode.com/svn/trunk/webrtc/

- mkdir /home/dhaval/qtrtc/qt-everywhere-opensource-src-5.0.1/qtwebrtc/src
- cp -rf /path of wbrtc code>/webrtc/ /home/dhaval/qtrtc/qt-everywhere-opensource-src-5.0.1/qtwebrtc/src/3rdparty

D. Creating /Modifying QT project files

QT uses .pro and .pri files to create make files at compile time. Project files contain all the information required by qmake to build application, library, or plugin. All the directories need be compiled should have an associate .pro file. There are Android.mk files which Google uses to compile WEBRTC for Android can be taken as reference to create .pro file. But instead of creating .pro file in individual directories I have decided to put all the .pro files as same location so that we can keep git version of code in cleaner manner and patching the latest changes/bug fixes available in webrtc can be easily ported. Following are the list of

.pro/.pri files which I created to compile webrtc as QT module.[12]

- qtwebrtc.pro
- src/src.pro
- src/webrtc/webrtc.pro
- src/3rdparty/rtp_rtcp.pro
- src/3rdparty/common_audio.pro
- src/3rdparty/libjingle_sound.pro
- src/3rdparty/cng.pro
- src/3rdparty/audio_coding_acm2.pro
- src/3rdparty/voice_engine_core.pro
- src/3rdparty/mock_video_coding.pro
- src/3rdparty/isacfix.pro
- src/3rdparty/media_file.pro
- src/3rdparty/desktop_capture.pro
- src/3rdparty/libjingle_media.pro
- src/3rdparty/audio_conference_mixer.pro
- src/3rdparty/audio_coding_module.pro
- src/3rdparty/g722.pro
- src/3rdparty/libjsoncpp.pro
- src/3rdparty/video_render_module.pro
- src/3rdparty/webrtc_video_coding.pro
- src/3rdparty/webrtc_vp8.pro
- src/3rdparty/rbe_components.pro
- src/3rdparty/audio_device.pro
- src/3rdparty/neteq4.pro
- src/3rdparty/isac.pro
- src/3rdparty/resampler.pro
- src/3rdparty/libjingle.pro
- src/3rdparty/3rdparty.pro
- src/3rdparty/libjingle_peerconnection.pro
- src/3rdparty/system_wrappers.pro
- src/3rdparty/libyuv.pro
- src/3rdparty/remote_bitrate_estimator.pro
- src/3rdparty/neteq.pro
- src/3rdparty/common_video.pro
- src/3rdparty/video_capture_module.pro
- src/3rdparty/ilbc.pro
- src/3rdparty/libjingle_p2p.pro
- src/3rdparty/pcm16b.pro
- src/3rdparty/video_coding_utility.pro
- src/3rdparty/libsrtp.pro
- src/3rdparty/webrtc_utility.pro
- src/3rdparty/webrtc_i420.pro
- src/3rdparty/video_engine_core.pro
- src/3rdparty/paced_sender.pro
- src/3rdparty/webrtc_opus.pro
- src/3rdparty/video_processing.pro
- src/3rdparty/bitrate_controller.pro
- src/3rdparty/g711.pro
- src/3rdparty/audio_processing.pro
- src/3rdparty/expat.pro

1. Integrating webrtc media APIs in QT

We need to modify following files taking as reference from Google chrome source code to integrate media (audio/video) APIs of webrtc in QT webkit browser. [13]

- qtwebkit/Source/WebCore/WebCore.pri
- qtwebkit/Source/WebCore/DerivedSources.pri
- qtwebkit/Source/WebCore/Modules/mediastream/UserMediaClient.h
- qtwebkit/Source/WebCore/Modules/mediastream/RTCPeerConnection.cpp
- qtwebkit/Source/WebCore/Modules/mediastream/RTCStatsResponse.cpp
- qtwebkit/Source/WebCore/page/Page.cpp
- qtwebkit/Source/WebCore/platform/qt/UserAgentQt.cpp
- qtwebkit/Source/WebCore/platform/mediastream/qt/MediaStreamCenterQt.cpp
- qtwebkit/Source/WebCore/platform/mediastream/qt/RTCPeerConnectionHandlerQt.h
- qtwebkit/Source/WebCore/platform/mediastream/qt/MediaStreamCenterQt.h
- qtwebkit/Source/WebCore/platform/mediastream/qt/RTCPeerConnectionHandlerQt.cpp
- qtwebkit/Source/WebCore/platform/Logging.cpp
- qtwebkit/Source/WebCore/platform/Logging.h
- qtwebkit/Source/WebCore/platform/graphics/qt/MediaPlayerPrivateMS.cpp
- qtwebkit/Source/WebCore/platform/graphics/qt/MediaPlayerPrivateMS.h
- qtwebkit/Source/WebCore/platform/graphics/MediaPlayer.cpp
- qtwebkit/Source/WebCore/Target.pri
- qtwebkit/Source/WebKit2/Target.pri
- qtwebkit/Source/WebKit2/WebProcess/WebCoreSupport/WebUserMediaClient.cpp
- qtwebkit/Source/WebKit2/WebProcess/WebCoreSupport/WebUserMediaClient.h
- qtwebkit/Source/WebKit2/WebProcess/WebPage/WebPage.cpp
- qtwebkit/Source/WebKit/WebKit1.pro
- qtwebkit/Source/WebKit/qt/WebCoreSupport/UserMediaClientQt.h
- qtwebkit/Source/WebKit/qt/WebCoreSupport/QWebPageAdapter.cpp
- qtwebkit/Source/WebKit/qt/WebCoreSupport/UserMediaPermissionClientQt.h
- qtwebkit/Source/WebKit/qt/WebCoreSupport/UserMediaClientQt.cpp
- qtwebkit/Source/WebKit/qt/WebCoreSupport/QWebPageAdapter.h
- qtwebkit/Source/WebKit/qt/WebCoreSupport/UserMediaPermissionClientQt.cpp
- qtwebkit/Source/WebKit/qt/WidgetApi/qwebpage_p.h
- qtwebkit/Source/WebKit/qt/WidgetApi/qwebpage.h
- qtwebkit/Source/WebKit/qt/WidgetApi/qwebpage.cpp
- qtwebkit/Tools/qmake/mkspecs/features/features.prf

We also need to create adaptation layer so that webrtc APIs can be readily integrated in QT webkit. Below are the files for webrtc/qt webkit adaptation…

- webrtc/api/api.pri

- webrtc/api/qrtcpeerconnection.h
- webrtc/api/qtwebrtcglobal.h
- webrtc/api/qrtcpeerconnection.cpp
- webrtc/api/qinclude.h
- webrtc/impl/impl.pri
- webrtc/impl/qwebrtcusermediafactory.cpp
- webrtc/impl/platform_video_capture.h
- webrtc/impl/webcamencoderfactory.cpp
- webrtc/impl/platform_audio_device.h
- webrtc/impl/qwebrtcusermediainterfaces.h
- webrtc/impl/qwebrtcusermediafactory.h
- webrtc/impl/qwebrtcusermediainterfaces.cpp
- webrtc/impl/qwebrtcusermediadecoderfactory.h
- webrtc/impl/webcamencoderfactory.h
- webrtc/impl/devicefactory.h
- webrtc/impl/qwebrtcusermediadecoderfactory.cpp
- webrtc/impl/qwebrtcusermediapluginfactory.cpp
- webrtc/impl/platform_video_capture.cc
- webrtc/impl/devicefactory.cpp
- webrtc/impl/qwebrtcusermediapluginfactory.h
- webrtc/impl/platform_audio_device.cc
- webrtc/webrtc.pro

E. Compiling QT with WEBRTC enabled

Below is the step to compile /build and install QT with webrtc[14]

- Configure QT with webrtc enabled…
  # cd /home/dhaval/qtrtc/qt-everywhere-opensource-src-5.0.1/
  # ./configure -opensource -v -confirm-license -prefix $PWD/../build -force-pkg-config -gstreamer -glib -icu -webrtc -optimized-qmake -no-accessibility -no-cups -no-qml-debug -no-xshape -qt-xcb -no-xsync -no-xvideo -no-xinerama -no-xcursor -no-xrandr -no-xfixes -no-xinput -no-xinput2 -no-xkb -no-nis -no-dbus -no-sql-db2 -no-sql-ibase -no-sql-mysql -no-sql-oci -no-sql-odbc -no-sql-psql -no-sql-sqlite -no-sql-sqlite2 -no-sql-sqlite_symbian -no-sql-tds -no-openvg -no-linuxfb
- Compile
  # make
- Install
  # make install

The above step will install QT in /home/dhaval/qtrtc/qt-everywhere-opensource-src-5.0.1/build directory. [14]

F. Debugging and verifying

QT is very large and includes thousands of files. To debug and understand the flow I used GNU gdb debugger.

GDB, the GNU Project debugger is very powerful and helpful to debug c/c++ code at source level. It allows seeing what is going on `inside' another program while it executes or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help catch bugs in the act:

- Start program, specifying anything that might affect its behavior.
- Make program stop on specified conditions.
- Examine what has happened, when program has stopped.
- Change things in program, to experiment with correcting the effects of one bug and go on to learn about another.

We can see function trace and symbols from dynamic library. Even if the library is not loaded at present, we can the breakpoint and run the program. When during execution, library gets loaded and function get executed, breakpoint will be hit and we can see the back-trace.[15]

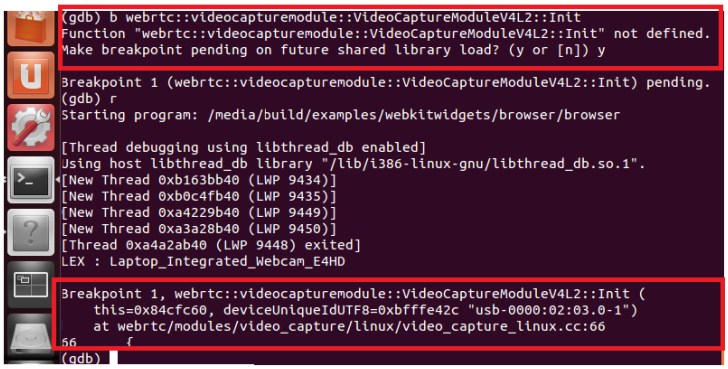

Fig.6

The QT browser can be started from command line…
#/home/dhaval/qtrtc/qt-everywhere-opensource-src-5.0.1/build/examples/webkitwidgets/browser/browser

It will open the QT browser with default page http://qt-project.org/

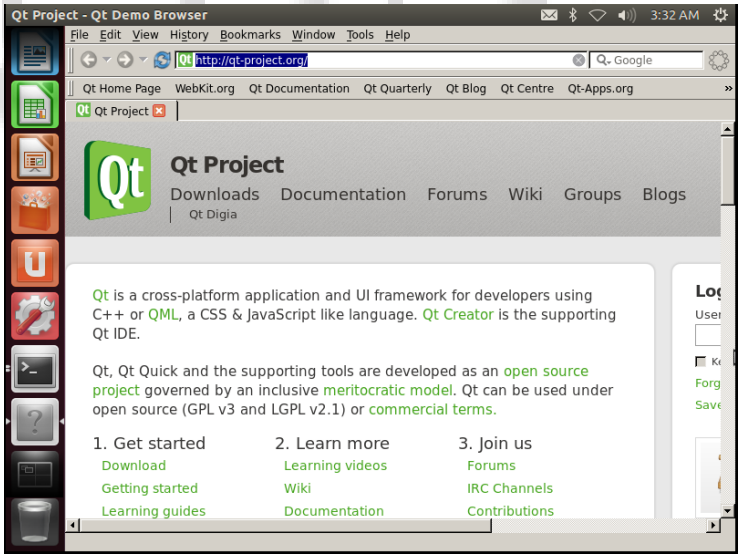

Fig.7

Now we can start RTC web applications which are available at http://www.html5rocks.com/en/tutorials/webrtc/basics/

VII. RESULT AND ANALYSIS

There are three APIs available in webrtc for html application…

A. MediaStream

Get access to data streams, such as from the user's camera and microphone.

B. RTCPeerConnection

Audio or video calling, with facilities for encryption and bandwidth management

C. RTCDataChannel

Peer-to-peer communication of generic data

In this project we have verified first two APIs that is getting user media and making audio/video call. We have not verifies data channel.

VIII. CONCLUSION AND FUTER WORK

Google has started many open source project to help embedded community. Webrtc is next to android which will change the whole business concept. In present there are very less applications to real time video communication and most of them are paid. After using webrtc in QT browser, many cheap and smart products can be made.

- Stage.1: In stage 1 we will try to implement QT browser to get media streams (Audio/Video) and identify it and run on it
- Stage.2 In stage 2 we will try to implement QT browser to communicate over other node with peer to peer connection on Media streams (Audio/Video) and identify it, connect it and run it.

ACKNOWLEDGEMENT

The finishing of this dissertation is an exciting achievement and has only eventuated through the ongoing support of many people who I now wish to acknowledge.

At first I would like to thanks to **God** who gives me the strength & blesses to enable me to live such a wonderful & beautiful life. At second I would like to thanks to my parents who appreciate me through my academic years of life with their love & supports.

Of course, I could never have reached this point without the great assistance & worm help & guidance of "Shankar Patel" who appreciates me to build this project and make a supportive guidance and coordination throughout this project.

REFERENCE

[1] http://www.html5rocks.com/en/tutorials/WEBRTC/basics/
[2] http://chimera.labs.oreilly.com/books/1230000000545/ch18.html
[3] http://ieeexplore.ieee.org/Xplore/home.jsp?reload=true
[4] http://www.computer.org/portal/web/computingnow/software
[5] http://www.softwaremag.com/Content/ContentCT.asp?P=1778
[6] http://www.WEBRTC.org/
[7] http://v4l-test.sourceforge.net/
[8] http://gstreamer.freedesktop.org/
[9] http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-good-plugins/html/gst-plugins-good-plugins-v4l2src.html
[10] http://wiki.oz9aec.net/index.php/Gstreamer_cheat_sheet
[11] http://qt-project.org/wiki/Building-Qt-5-from-Git
[12] http://qt-project.org/doc/qt-5/build-sources.html
[13] http://download.qt-project.org/official_releases/qt/5.2/5.2.1/single/qt-everywhere-opensource-src-5.2.1.tar.gz
[14] http://qt-project.org/doc/qt-4.8/qmake-project-files.html
[15] http://www.sourceware.org/gdb/

