WebRTC Applications and Uses in the Tech Industry

Kanika Anand, Ashley Love, Khoa Nguyen, Sindhu Prasad, Neeraj Sharma
San Jose State
University, online.kanika@gmail.com, Ashley.Love@sjsu.edu, trongkhoa@gmail.com, Sindhu.it@gmail.com, Neeraj.Sharma@sjsu.edu

Abstract - Technology based companies face a variety of problems, such as rapid deployment, cross-browser support, and scalability. By using WebRTC, a new web technology in market, we propose to solve these problems for a variety of industries, such as video streaming, VoIP. This paper discusses in depth about the WebRTC open source project, its APIs and the general architecture. Emphasis has been laid down to bring the best out of WebRTC latest technologies, and to enforce the easiest way of peer-to-peer communication. Analysis is made on what problems do WebRTC solve when compared to its existing peer technologies. Case studies are discussed in detail to get a better understanding of WebRTC in real world web applications. Later we put forward a web application idea called 'WeWatch', where friends can watch videos virtually. A prototype and basic architecture of our proposed system is presented and explained along with the technologies we are going to use in the development process. This paper elaborates on the concepts of WebRTC, its adaption in web applications and a proposed idea.

Index Terms – WebRTC, TURN, STUN, JESP, SDP.

INTRODUCTION

WebRTC Definition

WebRTC stands for Web Real-time Communications and is an open source project released by Google. It consists of a set of API's, which facilitates bidirectional, full-duplex communication between the browsers. It is analogous to making a phone call. Two browsers can exchange data, which could be in any form such as audio, video or multimedia. Data exchange can only happen if both the browsers are compatible with WebRTC. It is supported by most of the browsers including Chrome, Mozilla and Opera.

This technology is a free and device independent method to communicate across devices using browsers. It provides with a high quality of audio and video thus enriching the user experience without the use of any additional applications or servers. WebRTC provides peer-to-peer communication without the use of external plugins as it happens with Adobe Flash. More details about it are given in background section.

Overview of WebRTC

WebRTC communication is between two browsers and is not the traditional client-server communication. In the latter one, we know the server's address or end point and a request is targeted to that web server. In WebRTC, one peer initially doesn't have the IP and port to which it can start communicating with. It sends a request to a STUN (Session Traversal Utilities for NAT) server to retrieve its public-facing IP address. A NAT device is used for converting the private IPs to public facing IPs, through the use of mapping tables. Other peers can follow the same process to know their public IP addresses. Every peer generated a set of ICE (Interactive Connectivity Establishment) candidates, which are details about other peers in the network. Each candidate represents an IP address, a port and transport protocol to be used. Next step is to establish a network session connection with another peer. Session management and negotiation can take place using various protocols and the most famous among them is Session Initiation Protocol (SIP). It works with Session Description Protocol, which is an application layer protocol. An 'Offer' request is sent to the peers using a signaling protocol (e.g. SIP) and a suitable peer send back an 'Answer' response. The peer selects an optimal ICE candidate and establishes an active data channel between them.

Industry Applications

Most important application of WebRTC is for enterprise companies that have a consumer-facing web. Enterprise can render click-to-call or click-to-chat capabilities on their website. Customers can click and have their queries answered on a voice/ video call or over a chat conversation. The end users do not have to move to any additional applications or download any additional plugins to their browser. As WebRTC being an open source community, enterprises are using it for various applications. It could be a easy way to educate anyone landing on your website about the services/products and provide quick communication mode for customer service.

BACKGROUND

History of WebRTC

Historically, real-time communication has been very complex to achieve and required expensive infrastructure, technologies and licensing just to get started with. Integrating real-time technologies with existing applications has been a costly affair. Google desperately needed some real-time technologies to support its Google Talk service which used to work through plugins. In 2011, Ericsson built the first WebRTC prototype. WebRTC now implements standards and enables plugin-free real-time audio, video and data communication.

The need for WebRTC arises from the following limitations:

- Number of web services needs real-time communication, but the support for real-time communication comes from downloads, native apps or plugins.
- Downloads, installation and upgrades of different plugins in a browser is complex and unstable.
- Plugins, in general, offer complex deployment and difficult to debug and maintain. Most of the plugins require licensing and users do not prefer to install a new plugin for an application every time.

The motivation behind WebRTC is that the API's and functionalities offered by WebRTC should be open sources and easily accessible through a browser, without the need for installing and downloading different plugins. And it should be at least as efficient as existing technologies that offer real-time communication.

Existing Work

Real-time communication is not new. Landline telephone systems are here for decades now. Then with the advent of internet, VOIP(Voice over Internet Protocol) came into picture. Later with the development of new software plugins like Flash, Google chats the real time voice/video chat is done through our desktops and mobile phones. Today all these systems are in use and cannot be replaced in certain places. The latest innovation we have is WebRTC, which brings real time communication to our web browser with no hassle of extra effort to make it possible. The below section discusses various technologies PSTN, SIP protocol and Flash and the problems they posed and then how WebRTC solved them.

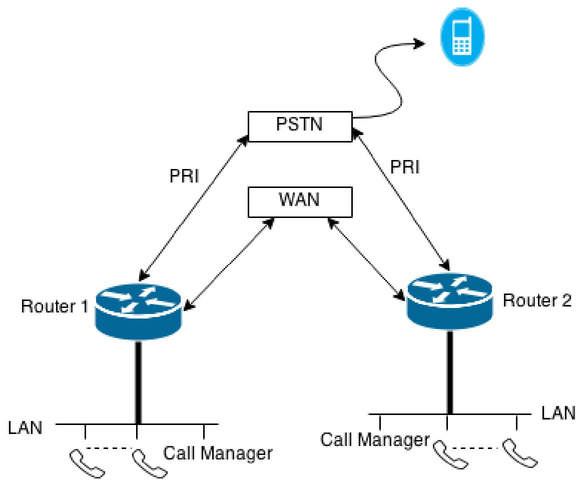
I. PSTN

PSTN is Public Switched Telephone Network (Legacy Telephone Service). These are traditional interconnected phone systems over which landline telephone calls are made via copper wires as analog signals. PSTN relies on circuit switching. To establish a phone connection to another, a phone call is routed through various switches functioning at various levels - local, regional and international. In an enterprise world, for example, Router 1 located at San Francisco branch and Router 2 located at New York branch are connected via WAN and for the outside world they are connected through dedicated PRI (Primary Rate Interface) lines which are limited to 23 concurrent calls as shown in Figure 1 below. For 24th call another PRI line should be set up which is expensive. PSTN is a legacy concept. In enterprise world, PSTNs are expensive and has limited scalability for communication.

II. SIP Trunking

Session Initiation Protocol (SIP) is a communication protocol for signaling and controlling sessions between two or multi parties for audio/video calls, interactive gaming. One of their popular applications is in VOIP systems. In SIP world, routers are no longer connected to physical circuit but

FIGURE 1
REAL-TIME COMMUNICATION THROUGH PSTN, WAN, LAN



are tied to the bandwidth of the channel provided by the SIP provider network as shown in the figure below. This is known as SIP Trunking. PSTN is accessed via SIP service. The number of concurrent calls accommodated depends on the bandwidth of the channel. Multiple calls are supported by adding an additional channel which can be done on the fly with less or no cost. In enterprise world, SIP protocol enables effective multimedia communication with scalability and low cost. Also, SIP supports any kind of applications - voice enabled or video enabled. Even though the client side applications changes in future, say from a simple application to video enabled application, same SIP channels can be used. On the other side, SIP enabled VOIP systems need softphones (a dedicated software program to make telephone over the internet)

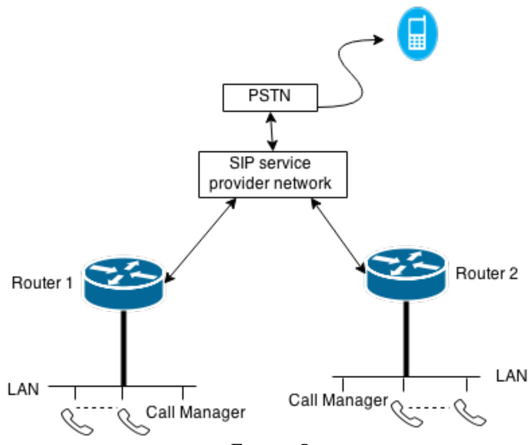


FIGURE 2
REAL-TIME COMMUNICATION VIA SIP SERVICE PROVIDER NETWORK [4]

III. *PSTN and SIP in WebRTC*

In WebRTC, control data, the session information is exchanged between the two browser clients via a web server. To fulfil this WebRTC can use any mechanism like Websockets, Google Cloud Messaging or XMLHTTPRequest (XHR). After this exchange, the data streams directly flow between the browser clients. To do this WebRTC can use any protocol like JSON, SIP/XMPP. This flow is shown in the figure below.

Clearly, WebRTC has many advantages over PSTN. WebRTC needs no dedicated connection lines, easily scalable and cost effective. Next is WebRTC vs SIP and WebRTC is not a refinement of SIP. Both have similarities as they provide real time communication via the Internet. SIP can function independently but WebRTC need some help in establishing connectivity. In fact WebRTC can use SIP as protocol. Inclusion of SIP makes WebRTC better. The advantages of WebRTC over SIP are: Unlike SIP, they need no softphones. Both can do voice/video calls and interactive gaming but WebRTC bring it on web browsers. Also WebRTC improves customer relations because it is web based.

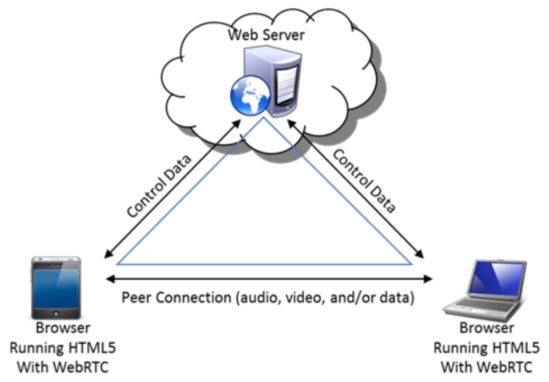

Web Server
Control Data
Control Data
Peer Connection (audio, video, and/or data)
Browser Running HTML5 With WebRTC
Browser Running HTML5 With WebRTC

FIGURE 3
REAL-TIME COMMUNICATION WITH WEBRTC [4]

IV. *Flash Plugins*

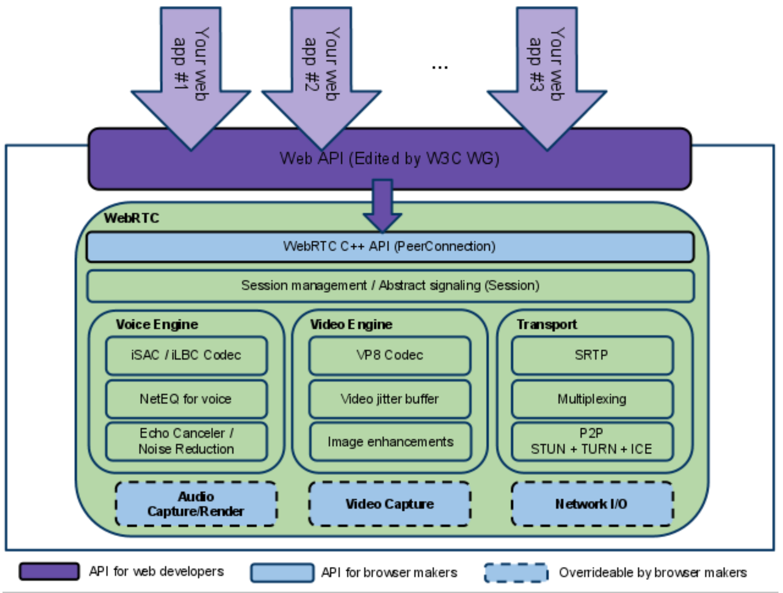
Plugin software extends browser capabilities. Adobe Flash plugin, commonly known as Flash, enables streaming of bidirectional audio/video content, delivering interactive media content on web pages and Flash-enabled software. For this, browsers need plugin. They need licensing to use in our web applications and also the browsers should have Flash player. This is an overhead to an application developer as well as the end user. WebRTC bring all the features that Flash could bring on the web browsers trouble-free. No plugin installation, no licensing with high quality.

ARCHITECTURE

The main architecture of WebRTC is shown in the following diagram. There are two main components of WebRTC engine, WebRTC C++ API and the Web API by W3C.

WebRTC C++ APIs

The internal WebRTC API's, offered by the WebRTC engine, are for browser developers. These API's are used by browser developers to create Web API's for web developers. WebRTC C++ API is the main implementation of WebRTC and all the components shown below the layer of WebRTC C++ API in the diagram, is the main work done by the WebRTC engine. Browser developers also use the last three overrideable components, which are Audio Capture/Renderer, Video Capture and Network I/O. These components are used to provide web developers with standard API's independent of browser and operating system the browser is running on. These standard API's are included in the Web API's and help web developers to



capture audio and video of the host running the browser.

FIGURE 4
WEBRTC ARCHITECTURE FROM WEBRTC.ORG

WebRTC essentially isolates web developers and browser developers from the communication hassles and complexities involved in peer-to-peer connection. The codes and protocols used by WebRTC engine ensures all the steps involved in setting up and maintaining real-time communication are successfully carried out without the browser and web application to carry out any task. Some of the tasks that are taken care of by WebRTC engine are:

- Hide nuances of packet losses during communication.
- Reduction and suppression of noise.
- Automatic gain and quality improvement.
- Adapting network communication to the bandwidth available.
- Cancellation of echo to improve communication quality.
- Relay server messaging if peer-to-peer connection fails.

- Resolution of public addresses behind NAT firewalls using STUN servers.

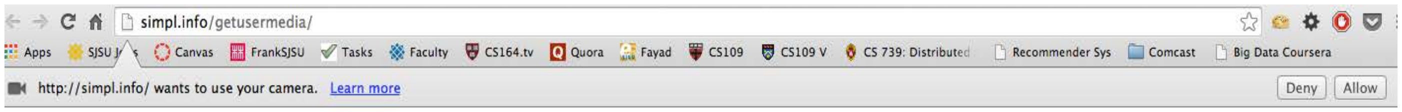
WebRTC APIs

WebRTC APIs are the API's offered by WebRTC engine which is built in the browser. These API's are used by the web developers for development of web applications that use WebRTC at the core. WebRTC offers three API implementations:

1. MediaStream (aka gUM or getUserMedia)
2. RTCPeerConnection
3. RTCDataChannel

MediaStream API is also known as getUserMedia(), since this is the name of the API offered by various browsers. MediaStream API is used for offering a synchronized stream of media. Referring to the architecture diagram, getUserMedia API essentially uses the last three components shown in dotted lines, namely, Audio Capture/Renderer, Video Capture and Network I/O. These three components are hooks offered by WebRTC engine to streamline and synchronize media streams across different browsers and across different hardware and operating systems combinations. WebRTC support is limited to some browsers and their specific versions. The main task of getUserMedia is to capture the user's audio and video with users' permission, irrespective of the browser, hardware and operating system user is running. The screenshot below shows getUserMedia API in action asking of users permission.

getUserMedia is available in the global scope of the page. Access to getUserMedia can be received by just calling getUserMedia or navigator.getUserMedia.



http://simpl.info/ wants to use your camera.

FIGURE 5. GETUSERMEDIA() REQUESTING PERMISSION FROM USER

getUserMedia expects three parameters which are Constraints, Success callback function and Error callback functions.

Constraints Object: Constraints object is used for setting different aspects and resolution of the video stream being received from user's media or getUserMedia. Current support for constraint object intends to tweak aspect ration, facing mode (front or back camera), frame rate, height and width of video stream. For example, for playing HD video the following object can be passed as constraints:

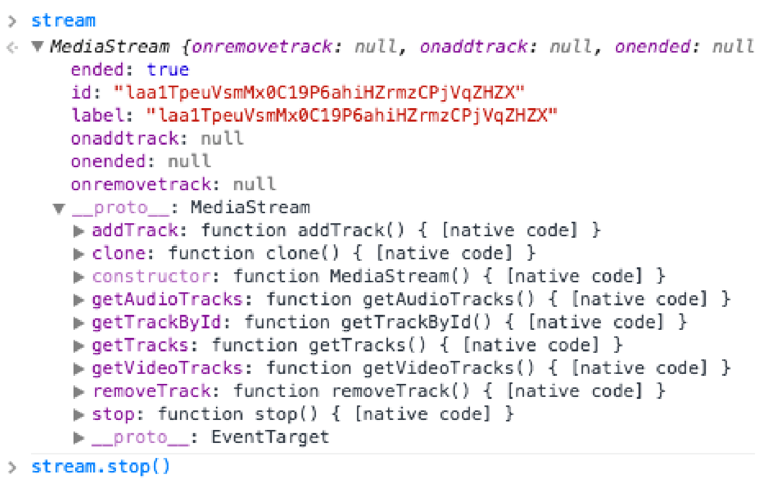
```
var hdConstraints = {
 video: {
   mandatory: {
     minWidth: 1280,
     minHeight: 720
     }
   }
 };
```

Or for getting access to only audio of user's media:

```
var audioConstrations = {
 audio: true
}
```

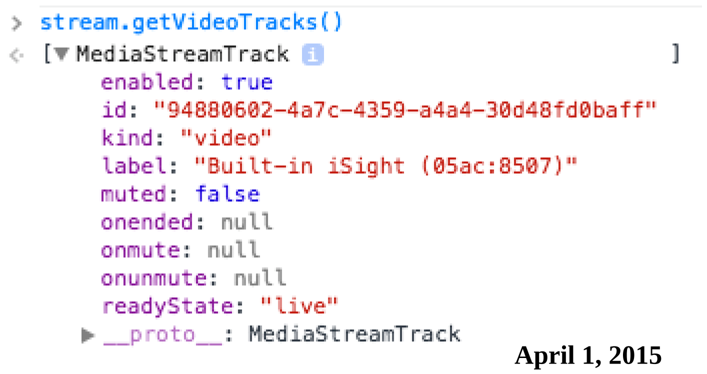
Success callback: This function is passed a MediaStream object if getting access to user media is successful. A MediaStream object can then be used to take actions on user media like audio or video. The following screenshot displays the MediaStream object received once user accepts the permissions to grant access to media and the video plays back.

FIGURE 6. MEDIA STREAM CODE SNIPPET



```
> stream
< ▼MediaStream {onremovetrack: null, onaddtrack: null, onended: null
    ended: true
    id: "laa1TpeuVsmMx0C19P6ahiHZrmzCPjVqZHZX"
    label: "laa1TpeuVsmMx0C19P6ahiHZrmzCPjVqZHZX"
    onaddtrack: null
    onended: null
    onremovetrack: null
  ▼ __proto__: MediaStream
    ▶ addTrack: function addTrack() { [native code] }
    ▶ clone: function clone() { [native code] }
    ▶ constructor: function MediaStream() { [native code] }
    ▶ getAudioTracks: function getAudioTracks() { [native code] }
    ▶ getTrackById: function getTrackById() { [native code] }
    ▶ getTracks: function getTracks() { [native code] }
    ▶ getVideoTracks: function getVideoTracks() { [native code] }
    ▶ removeTrack: function removeTrack() { [native code] }
    ▶ stop: function stop() { [native code] }
    ▶ __proto__: EventTarget
> stream.stop()
```

In console, stream.stop() then stops the media play back being played on the screen. Notice that MediaStream object has an "id" and a "label". getVideoTracks and getAudioTracks are observed to be functions offered by MediaStream object, which essentially return an array of MediaStream tracks which define the type of camera or device, current status of media stream, the type of stream, if the device is muted or not etc. The following screenshot



```
> stream.getVideoTracks()
< [▼MediaStreamTrack i                                   ]
     enabled: true
     id: "94880602-4a7c-4359-a4a4-30d48fd0baff"
     kind: "video"
     label: "Built-in iSight (05ac:8507)"
     muted: false
     onended: null
     onmute: null
     onunmute: null
     readyState: "live"
   ▶ __proto__: MediaStreamTrack
```

shows MediaStream tracks of FrontCamera of MacBook pro.

FIGURE 7 VIDEO TRACK CODE SNIPPET

To use this stream in an HTML page, it can be converted to a URL Blob using URL.createObjectURL() method supported in Chrome. This method returns a blob URL which can be then set as source of a video tag in HTML page. An example of a Blob URL is:

```
blob:http://localhost/c745ef73-
ece9-46da-8f66-ebes574789b1
```

Blob URL's are unique and are created by the browser for the lifetime of the application. Essentially, this consumes memory and storage if an application is creating lot of blob URLs and not freeing up the storage. A good practice is to revoke the URL after use by calling revokeObjectURL(blobURL).

MediaStream API is planned to synchronize the access to user media, irrespective of the type of media, be it disk, online, compact disc or else.

Error callback: If the user does not allow access to media, this callback function is provided an error object and then it can be used for fallback mechanism.

RTCPeerConnection API: Peer connection API is the API of WebRTC that handles streaming data communication between two peers and ensures communication quality, stability and efficiency. There are two different API's implemented for RTCPeerConnection, which is webkitRTCPeerConnection by Chrome and mozRTCPeerConnection by Mozilla. Once the standards are finalized, these experimental API's will be stable and prefixes will be removed. RTCPeerConnection helps to establish a connection between two browsers or peers and then maintains a healthy communication between the two peers. The communication can use any media like files, audio and video till it is provided as a stream to peers. RTCPeerConnection API implementation handles code conversions and protocol adaptations underneath.

RTCPeerConnection API's are best explained through an example. This example includes a caller (local) and a callee (remote). The example given below does not include signaling and assumes local and remote callers are on the same page, to provide simplicity for explanation. The following steps are taken by caller and callee to establish a peer connection.

Caller Steps
1. Create a new RTCPeerConnection and add the stream, received from getUserMedia to this connection using addStream method.

```
// servers is an optional config file
//(see TURN and STUN discussion below)
pc1 = new
webkitRTCPeerConnection(servers);
// ...
pc1.addStream(localstream);
```

2. Create an offer and set session description. For pc1, which is local peer connection, session description being set will be local. For pc2, which is remote peer connection, session description being set will be remote. createOffer is the method offered by RTCPeerConnection to create an offer.

```
pc1.createOffer(gotDescription1);
//...
function gotDescription1(desc){
 pc1.setLocalDescription(desc);
 trace("Offer from pc1 \n" +
desc.sdp);
 pc2.setRemoteDescription(desc);
 pc2.createAnswer(gotDescription2);
}
```

Callee Steps
1. Listen for onaddstream event offered by RTCPeerConnection. Once, caller, which is pc1, adds a stream to the peer connection, use this stream to display it on the webpage.

```
pc2 = new webkitRTCPeerConnection(servers);
pc2.onaddstream = gotRemoteStream;
//...
function gotRemoteStream(e){
 vid2.src = URL.createObjectURL(e.stream);
}
```

RTCDataChannel API: As mentioned in the previous discussions, RTCPeerConnection does not only support audio and video, but supports any arbitrary data that can be sent through the channel that is set up through RTCPeerConnection. RTCDataChannel API enables this functionality. WebRTC engine ensures low latency and high throughput for any sort of communication, so this API is useful for large file transfers between two browsers. RTCDataChannel API uses RTCPeerConnection for setting up the channel for P2P communication and offers the following features.

- P2P session setup using RTCPeerConnection.
- Arbitrary data communication, audio, video, files, text etc.
- Configurable delivery semantics for reliable and unreliable, offering UDP or TCP like communication.
- Enables support for multiple channels running simultaneously.

Following is an example of using RTCDataChannel API setting up a session using RTCPeerConnection.

```
var pc = new
webkitRTCPeerConnection(servers,
         {optional: [{RtpDataChannels:
        true}]});

        pc.ondatachannel = function(event) {
         receiveChannel = event.channel;
         receiveChannel.onmessage =
        function(event){
           document.querySelector("div#receive
        ").innerHTML = event.data;
         };
        };

        sendChannel =
        pc.createDataChannel("sendDataChannel"
        , {reliable: false});
document.querySelector("button#send").onclick
= function ()
{
        var data = document.querySelector
                ("textarea#send").value;
        sendChannel.send(data);
};
```

A new RTCPeerConnection instance is created with options RtpDataChannels as true. Ondatachannel is an event supported by RTCDataChannel API and when triggered, indicates an event created on the channel being established. Channel is created using createDataChannel. The events that are triggered on the channel include ‘onmessage’, which indicates a received message on the channel. Data can be sent using send method. Naming conventions are similar to websockets. The data received and sent can be used by the application similar to websockets. Though, RTCDataChannel is expected to be much faster than websockets because of the low latency in peer-to-peer connection. In worst-case scenario, a relay server will be added to the peer-to-peer connection between browsers using RTCDataChannel, however, the performance will still be comparable or better than websockets.

Signaling in WebRTC
WebRTC API’s do not offer any signaling mechanisms natively. To establish a peer-to-peer connection between two remote servers, there is a need for exchanging some metadata initially. The need for signaling messages thus arises. WebRTC does not limit the developers for use of any technology for writing signaling mechanisms. During our research, we observed that websockets is the most favored and efficient way to write a signaling server for WebRTC applications. Signaling messages, essentially, include control messages, metadata information and messages to coordinate for communication. WebRTC does not specify any protocols for signaling.

Types of information exchanged using signaling:

- Session control messages to initiate communication, to stop communication and to report errors.
- Network configuration messages exchanging IP addresses, ports and other network configuration required for setting up peer-to-peer connection.
- Media capabilities that identifies codes and protocols supported by each peer to set up a common language to talk.

Signaling is used to establish a call or a peer-to-peer connection. Before using RTCPeerConnection or RTCDataChannel to send messages across to peers, signaling is required to establish a channel.

Let’s take an example to explain how signaling should work to establish a WebRTC call. Assuming A and B want to communicate, the following steps need to be taken to establish a peer-to-peer connection.

1. A creates a new instance of RTCPeerConnection and defines a handler for one candidate. WebRTC uses ICE frameworks for locating peers.
2. As soon as a candidate becomes available and ICE locates it, this handler is initiated.
3. A sends serialized data to B through the signaling channel, which is websockets.
4. As B receives a candidate message from A, B uses addIceCandidate to add A to remote peer description of RTCPeerConnection.

WebRTC clients, like A and B in the example above, also need to exchange metadata about their media streams, such as the codes supported, the resolution expected, frame rate etc. This information should also be exchanged to set up a compatible talk between the two peers. It is recommended to use Session Description Protocol (SDP) or JavaScript Session Establishment Protocol (JSEP), which complies to SDP. The common pattern to exchange meta information is to follow an offer and answer model. Following steps are taken by A and B to exchange this information:

1. A to create an offer by using createOffer of RTCPeerConnection and set local description of RTCPeerConnection to it’s local session which is a RTCSessionDescription object. This uses setLocalDescription method as described earlier.
2. A sends the session description to B through the signaling channel. As B receives the new candidate session description, it sets this description to remote description using setRemoteDescription. So, A’s description becomes B’s remote description and B’s description becomes A’s remote description.
3. B creates an answer to the offer using createAnswer method and passes in it’s remote description (A’s local description). A receives this message and sets it’s remote description to B’s local description.

These steps then establish a peer-to-peer connection successfully. It is required to have completed signaling before starting audio or video stream through WebRTC API's.

Signaling example code snippet:

```
var signalingChannel =
createSignalingChannel();
var pc;
var configuration = ...;

// run start(true) to initiate a call
function start(isCaller) {
   pc = new
RTCPeerConnection(configuration);

// send any ice candidates to the
//other peer
   pc.onicecandidate = function (evt)
{
       signalingChannel.send(JSON.stri
ngify({ "candidate": evt.candidate
}));
   };

// once remote stream arrives, show it
//in the remote video element
   pc.onaddstream = function (evt) {
       remoteView.src =
URL.createObjectURL(evt.stream);
   };

// get the local stream, show it in
//the local video element and send it
   navigator.getUserMedia({ "audio":
true, "video": true }, function
(stream) {
       selfView.src =
URL.createObjectURL(stream);
       pc.addStream(stream);

       if (isCaller)
           pc.createOffer(gotDescript
ion);
       else
           pc.createAnswer(pc.remoteD
escription, gotDescription);

       function gotDescription(desc) {
           pc.setLocalDescription(desc
);
           signalingChannel.send(JSON.
stringify({ "sdp": desc }));
       }
   });
}

signalingChannel.onmessage = function
(evt) {
   if (!pc)
       start(false);

   var signal = JSON.parse(evt.data);
   if (signal.sdp)
       pc.setRemoteDescription(new
RTCSessionDescription(signal.sdp));
   else
       pc.addIceCandidate(new
RTCIceCandidate(signal.candidate));
};
```

A reference of JSEP architecture is show in FIGURE 8. JSEP ARCHITECTURE:

The diagram shows that SessionDescription objects are are exchanged between Caller and Callee indicated by browsers through Signalling mechanism. Though, the media exchange (audio/video) is only between browser and browser.

WebRTC enables this media communication through STUN/TURN servers for NAT traversals, ICE framework for connecting to peers. ICE is a framework for connecting to peers directly. Initially, ICE tries to connect to peers using UDP and that involves resolving peers behind NAT firewalls using STUN servers. If UDP fails, ICE tries TCP, first http and then https. In some cases, when ICE is unable to locate peers behind enterprise firewalls and UDP fails, ICE will fall back to TURN serves acting as relay servers to transmit the messages using TCP. A diagram below shows the details of this communication.

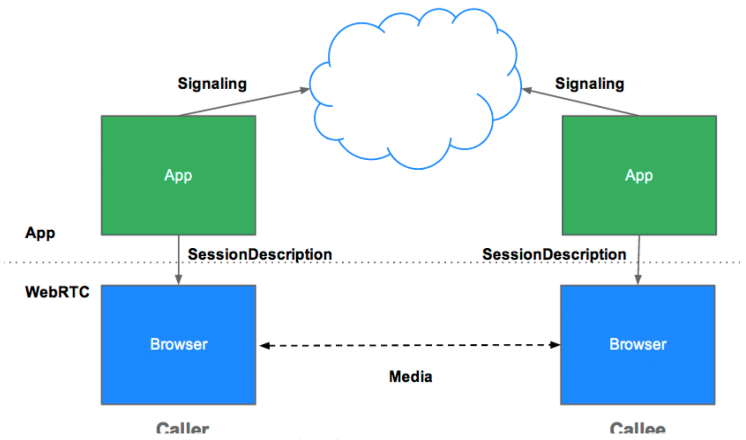
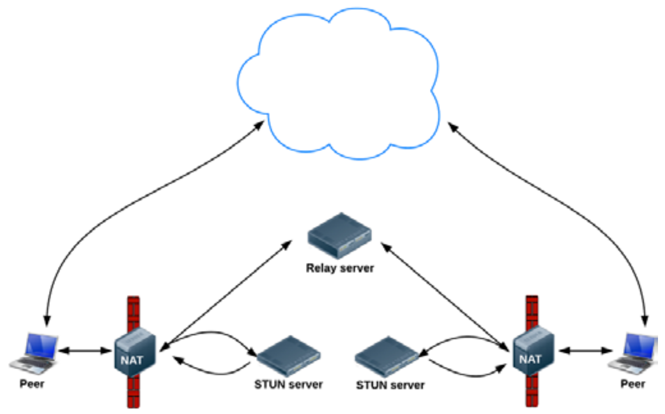


FIGURE 9. JSEP ARCHITECTURE

FIGURE 8. JSEP ARCHITECTURE



ADVANTAGES AND DISADVANTAGES OF WEBRTC

WebRTC, the technology that allows real time communication independent of browser or device, is a

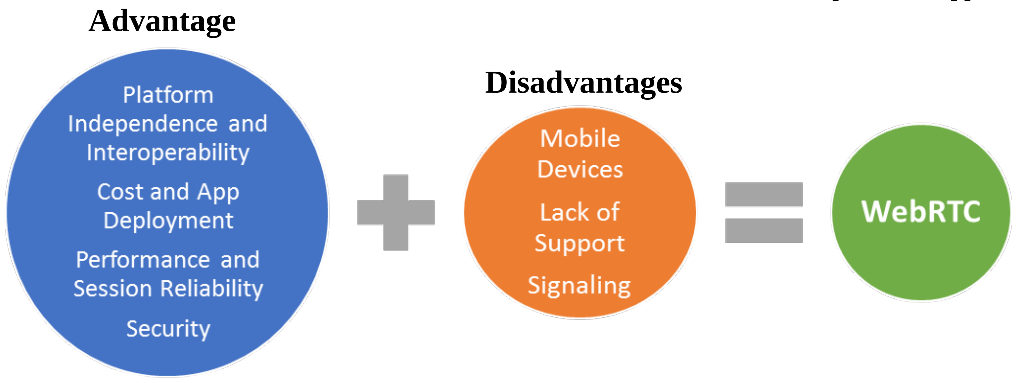


FIGURE 10.
ADVANTAGES AND DISADVANTAGES OF WEBRTC

communication medium that leverages a variety of benefits and overcomes many of the disadvantages of other technologies discussed earlier (i.e. flash and PSTN). The following concepts or features are discussed in the subsequent sections to elucidate the capabilities of webRTC: platform and device independence, information security, cost, network constraints, session reliability, application deployment, and interoperability [1] [2].

Platform Independence and Interoperability

Platform and device independence is a central issue that concerns developers, the system's functionality, user interface, and user experience. All of these problems tie into high level issues, such as inherent difficulties for hybrid and native architectures, maintenance, debugging, interface standards, performance, and other areas [3] [4]. Before the advent of tablets, PDAs, and other personal tech devices, the cross-platform issue was more straightforward, or at least there were fewer devices to consider. End users desire a consistent intuitive user experience, whereas developers want to have a single piece of code to be represented cross-platform without limited features or platform dedicated code. WebRTC is a solution to cross-platform problems and facilitates platform and device independence. To illustrate this idea, see Figure XX and XX.

FIGURE 11 represents the status quo system architecture of how cross-platform applications are supported; Figure 2, shows the difference in component interaction as a result of webRTC. As a result of the modified architecture, a portion of the platform specific code will be minimized for media and data. WebRTC is the architecture commonality between the different browsers or platforms; this technology streamlines, simplifies, and reduces the cost of the development process [1] [2] [4]. The standardized Web APIs and protocols, developed by W3C and IETF, are created to support cross-platform code writing, for mobile devices and

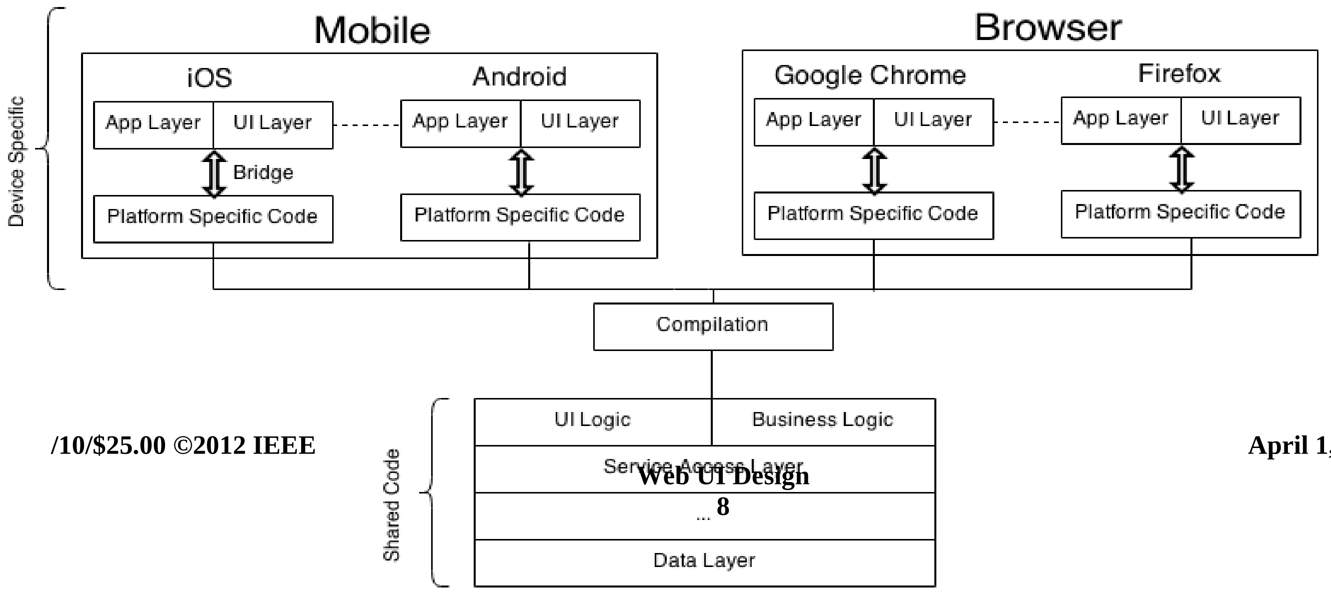


FIGURE 11. LEGACY ARCHITECTURE BEFORE WEBRTC

browsers, using the Web API for webRTC, and C++ based API, respectively.

Interoperability is one of the largest advantages and driving force behind developing WebRTC. The technology can allow different devices to connect, ranging from browsers to TVs. A diagram from Dialogic, a company that conducts surveys and provides WebRTC technology solutions, published XXXXXXXX to show different use cases for WebRTC:

The above figure encapsulates the primary scenarios where WebRTC can be used (ie. VoIP, Call Centers, Conferences). The cases operate optimally as a result of centralized signaling and fully or tightly-coupled media, [6]. The subsequent sections, which include case studies and a high-level prototype of systems, are explained in detail later.

Cost and Application Deployment

Cost is a major factor in any software product with a large portion coming from maintenance. A study published by Cambridge University and Gartner Research estimates that the amount of money spent on software upkeep is approximately 312 billion globally and can rise to 1 trillion in 2015 [5]. WebRTC is not a comprehensive solution to solving the technical debt dilemma, but can significantly minimize the probability of bugs by the ease of integration and deployment for WebRTC technology and associated protocols.

In order to effectively use video, voice, streaming ad other communication technology, many companies outsource the integration and setup of UC (Unified Communications) platforms [6]. The time and money for hiring a UC vendor far exceeds the cost for a developer to learn and use WebRTC. The technology is adaptable, flexible, and can accommodate existing technologies. With an open source community that supports the development of WebRTC and the plentiful code samples, WebRTC is a viable option for startups and large corporations. Servers and other hardware can be virtualized, there are no licensing fees, and the startup costs are nothing. WebRTC fulfills all budget ranges and provides a clear-cut platform independent solution for development.

Performance and Session Reliability

The performance is optimized by using the three main APIs: RTCPeerConnection, RTCDataChannel, and MediaStream. The table below outlines the performance benefits, which are nested in the APIs.

TABLE 1 API'S WEB PERFORMANCE BENEFITS [7]

API	Web Performance Benefits
RTCPeerConnection	• Find most effective P2P route (i.e. NAT, firewalls • Manage bandwidth
RTCDataChannel	• Low latency • Reliable
MediaStream	• Flexibility to be synchronous or not synchronous

Furthermore, performance degradation due to file transfers is circumvented because files no longer have to be downloaded onto a device. A user can look at another user's video, data, or other media by using WebRTC APIs.

In case there are session reliability issues or other problems that require debugging, the technology has a tool, WebRTC Internals, which can be used for remedying faults or anomalies. Information such as packet loss, bandwidth, event type, and other metadata can be filtered and processed for analysis.

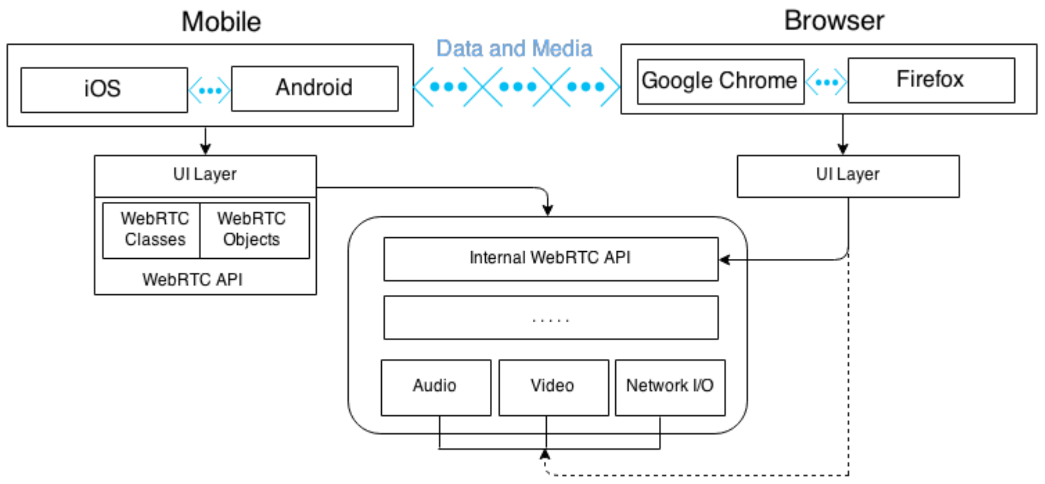


FIGURE 13. ARCHITECTURE AFTER WEBRTC

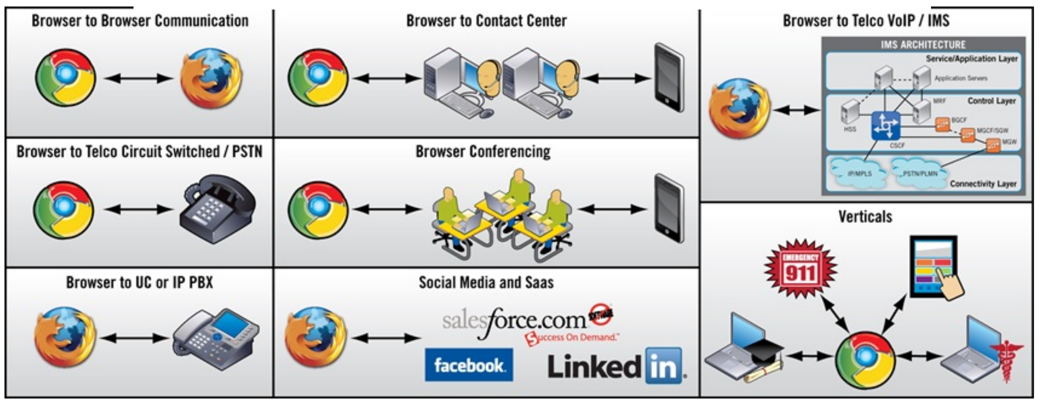

Browser to Browser Communication
Browser to Contact Center
Browser to Telco VoIP / IMS
IMS ARCHITECTURE
Browser to Telco Circuit Switched / PSTN
Browser Conferencing
Verticals
Browser to UC or IP PBX
Social Media and Saas
salesforce.com
facebook
LinkedIn

FIGURE 12. WEBRTC USE CASE DIAGRAM FROM DIALOGIC CORPORATION

Security

WebRTC is much more secure than other data and media transmission methods. By using HTTPS, devices or platforms send AES (Advanced Encryption Standard) data which the user must opt-in to receive or send [7] [10]. This reduces the chance of malware downloads posing as critical components for video viewing or browser stability. WebRTC creates a sandbox that impedes a hacker's ability to corrupt systems, which is a significant advantage. A Forbes Magazine article, titled "Future Crimes: Tech Threats from Hackers, China, Google, and Facebook," estimates that there are approximately 200,000 new malicious software programs introduced into the web every day; “antivirus software catches about five percent, and Verizon says that the average time to detect a data breach is 210 days [8].” By using an encrypted sandboxed technology and transmitting over HTTPS, SRTP (for audio and video), and DTLS (for video), the damage and risk of security breaches are significantly reduced. Beyond the ease of development and increased functionality, WebRTC aims to decrease the $375 to $575 billion dollars lost annually to hackers [9].

WebRTC Disadvantages

Even though this technology offers many benefits, there are downsides, as with any technology. The main concerns involve mobile devices, reliability, and support from Microsoft and Apple [3] [6] [9]. Mobile devices have strict performance constraints, due to processor and size limitations. Optimizations in the API or platform specific code may have to be developed in the future to account for these issues.

Call centers have reported reliability issues, but with the advent of using TURN and STURN servers in conjunction with WebRTC, some of these problems may now be irrelevant. Lastly, Microsoft and Apple, which are sizable contributors in the community, have not put their support behind this technology. Due to the lack of widespread support, this can thwart development and progress. Many standardizing bodies (i.e, W3C and IETF), have members from top companies, such as Microsoft and Apple. Without certain companies that represent a set of technologies, this impact the viability and direction of WebRTC.

Signaling, the way communication initiations occur, can have benefits and disadvantages. The session is established using signaling, but the protocols and mechanisms are not standardized, which can create issues for portability and maintenance. The session description can be very complex. Due to the density, the WebRTC was designed to not need certain features and have embedded defaults set [7]. Fortunately this is a feature that is not needed for many applications. Below is a code snippet that illustrates the complexity from Google’s I/O Conference in 2013:

```
v=0
o=- 7614219274584779017 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE audio video
a=msid-semantic: WMS
m=audio 1
RTP/SAVPF 111 103 104 0 8 107 106 105 13 126
c=IN IP4 0.0.0.0
a=rtcp:1 IN IP4 0.0.0.0
a=ice-ufrag:W2TGCZw2NZHuwlnf
a=ice-pwd:xdQEccP40E+P0L5qTyzDgfmW
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=mid:audio
a=rtcp-mux
a=crypto:1
AES_CM_128_HMAC_SHA1_80 inline:9c1AHz27dZ9xPI
91YNfSlI67/EMkjHHIHORiClQe
a=rtpmap:111 opus/48000/2
…
```

FIGURE 14. RTCSESSIONDESCRIPTION EXAMPLE FROM GOOGLE I/O CONFERENCE [11]

CASE STUDIES AND PROTOTYPES

A. *Video Conferencing System Based on WebRTC for Seniors*

I. *Purpose:*

There are many video and chat tools but they rarely support for elder people. The authors propose a way that can help elders video social networking with their relatives and friends easily through television which they already use everyday. This helps users to eliminate the learning curve of learning new technology. They can use the remote controllers to connect and talk to other people without understanding the technology.

II. *Technology Stack*

Authors propose to create web app with HTML5 and WebRTC. WebRTC provides two main API which are MediaStream and RTCPeerConnection. MediaStream provides API to access input stream. RTCPeerConnection allows to read the data from the output of MediaStream.

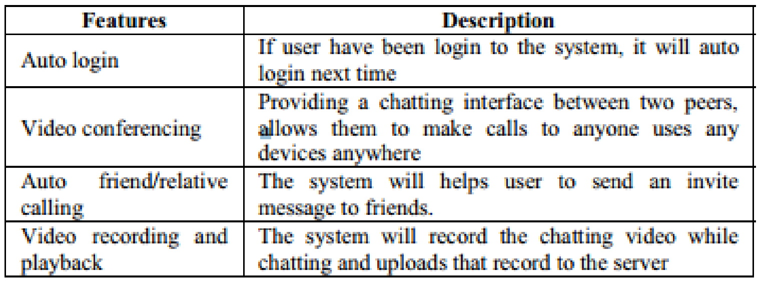
III. *System Overview*

The proposed system architecture is very close to the architecture of WebRTC where the server creates session for both parties connecting to each other. The graphical user interface is designed to accommodate lack of understanding and cut the learning curve of elders. The call initiation is reduced to two steps: First time Login and Send Request.

The goal of the system is to reduce the number steps required to activate the video call from elders to others. The system provides many features to support this such as Auto Login, Video Conferencing, Auto friend/relative calling, Video recording and playback.

IV. *Result*

The author use SUS (System Usability Scale proposed by John Brooke) to verify the usability and rating the web app. They invited 14 elders with different experiences with technology to experiment. Based on the questionnaires, the collected statistics is the following tables



Features	Description
Auto login	If user have been login to the system, it will auto login next time
Video conferencing	Providing a chatting interface between two peers, allows them to make calls to anyone uses any devices anywhere
Auto friend/relative calling	The system will helps user to send an invite message to friends.
Video recording and playback	The system will record the chatting video while chatting and uploads that record to the server

FIGURE 15. TABLE FROM VIDEO CONFERENCING PAPER SHOWING RESULTS

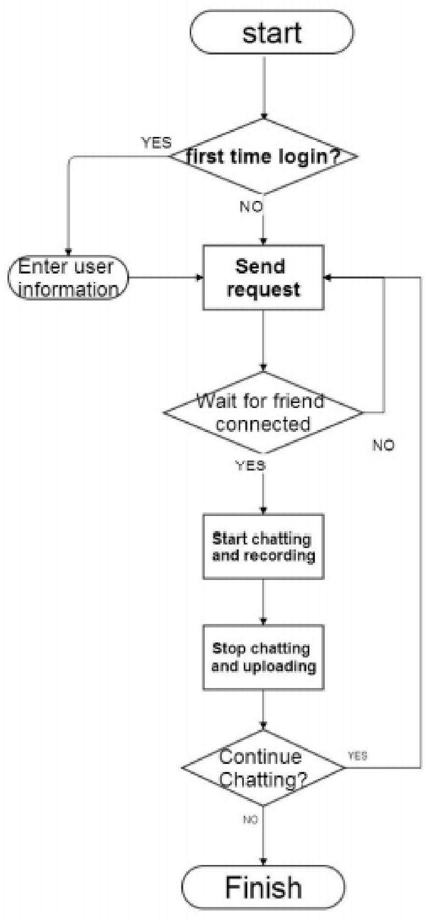


FIGURE 16. SYSTEM WORKFLOW FOR ELDER

According to Aaron Bangor, the SUS score is acceptable above 70, better from high 70 to upper 80, commercial system must above 90. So the experiment yields "acceptable" result.

II. Conclusion

There are not so many video chatting support seniors use. Through the research experiment, most users rate the proposed system was easy to use and acceptable for future development.

B. An On-Demand WebRTC and IoT Device Tunneling Service for Hospitals

I. Purpose:

The paper demonstrates the implementation of WebRTC gateway service that can route RTP traffic from a browser on a local hospital network to another browser on a local home network. The data transmits through port 80 only as restrictions of firewall and network security policy. It was used to tunnel sensor and control data for medical equipment at patient's house.

II. Technology Stack

The solution is built on top of HTML5 and WebRTC infrastructure. It provides three main services: a tunnel(TUN), a local RTC(Real-time Communication) gateway (LGAT) and a central RTC gateway(CGAT). LGAT services is deployed on local network behind firewalls while CGAT service is deployed on the public Internet. TUN mixes the traffic into a single well-known port and send it to another tunnel peer.

III. System Overview

The system is built on top of WebRTC and STUN. The traffic is routed and modify through each phase of finding and connecting two parties.

- Phase I: Offer and Answer Exchange
- Phase II: Candidate Exchange

The candidate exchange may be happened at before or after the phase 1. The authors solve the problem by caching results on both parties. When the complete information of both parties is received, the system triggers to the next step. This phase uses the gateway service heavily.

The candidate exchange is the first step to reroute the WebRTC traffic to the tunnel (TUN). In order to receive valid data back, the system needs to respond to the STUN requests correctly. The diagram below illustrates the reply to STUN requests.

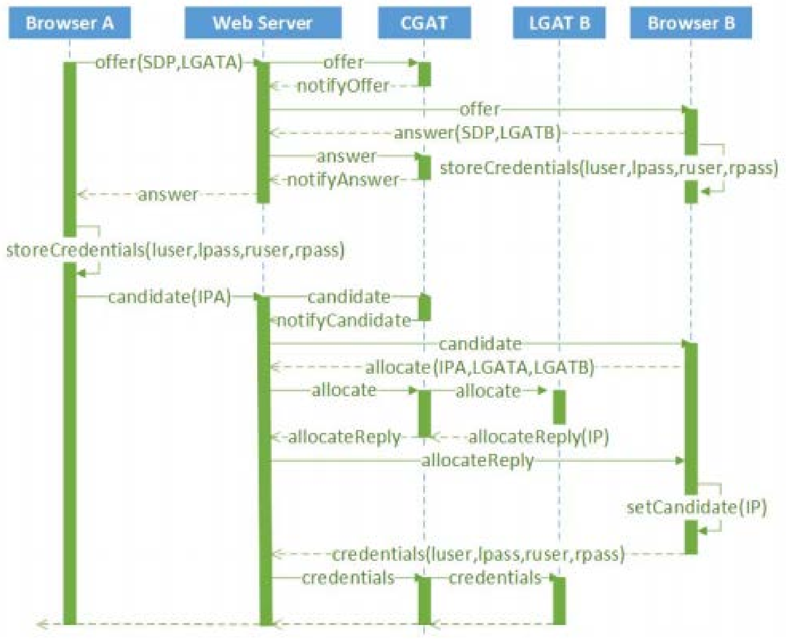


FIGURE 17 OFFER/ANSWER AND CANDIDATE EXCHANGE UML DIAGRAM [8]

- Phase III: STUN Endpoint Verification

Since the web browser will send multiple STUN requests to the gateway, it may halt the gateway to stop working. The solution is to ignore a certain number of requests before answering them correctly.

Phase IV: Data Traffic Routing

The gateway is ready to tunnel the data between the two browsers at phase IV.

IV. Result

- Latency Evaluation

Sandholm et al setups three scenarios one with fast laptops with Windows Experience Index(WEI) at 5.6, slow laptops with WEI at 3.2, wired and wireless network. They compared based on two metrics round-trip-time(RTT) and Jitter. The results yielded as below figures

Table xx: Summary of Experiment Values [4]

TABLE I. SUMMARY OF EXPERIMENT VALUES.

laptop	network	rtt			jitter		
		local gateway	remote gateway	direct	local gateway	remote gateway	direct
slow	wired	47	55	15	11	55	5
fast	wired	36	32	3	8	32	1
fast	wireless	48	94	25	12	94	7

According to the results, the slight lag may be acceptable but in some cases there is no stream at all. The local gateway on fast laptop yields the best result since it prevents cross host traffic.

- **Session Initiation Overhead**

The overhead is happened due to extra redirections and allocations for tunneling data between parties. The authors measure time from the moment users click on a button to initiate the call to the Phase I completed and to Phase IV start time. The getStats API of PeerConnection is used to determine when data on both sides are sent and received. The local gateway on residing on laptop yields the highest result. It can be explains that the cross traffic is reduced.

- **Scalability**

The gateway is designed to be shared among clients on the same local network. Therefore, it should be easy to reuse and handle concurrent streams. According to Sanholm et al, the video starts to feel like lagging at 200ms RTT. So the gateway under the condition of experiment environment can handle up to 10 concurrent session. This could be improved better performance if deploying onto cloud where the gateway will run on multi-CPU parallel servers.

V. Conclusion

The authors had a solution for tunneling WebRTC data plane traffic through strict firewall network. The gateway services can multiplex traffic from concurrent streams without leaking traffic and successfully route data to web port 80. The gateway services can be scaled by load-balancing and networking with other gateways on the same network to divide the load.

C. WeWatch Webapp Prototype

This is one of our project ideas, which would complement our research as well as give us an opportunity to implement a new idea. We propose to build a new web application using which friends can watch a video together.

The above diagram shows a basic architecture of the web application. Assuming four friends are connected and Friend A wants to play a video to watch with three of his other friends. We plan to setup a node server using socket.io for signalling mechanism for setting up this call. We plan to stream the video playback through the RTCPeerConnection and arbitrary control messages of video play/pause through RTCDataChannel. And switch to the user's audio to talk if the video is paused. We attempted to build a prototype of one page application which streams the webcam to the same page, for simplicity, and a screenshot is shown below:

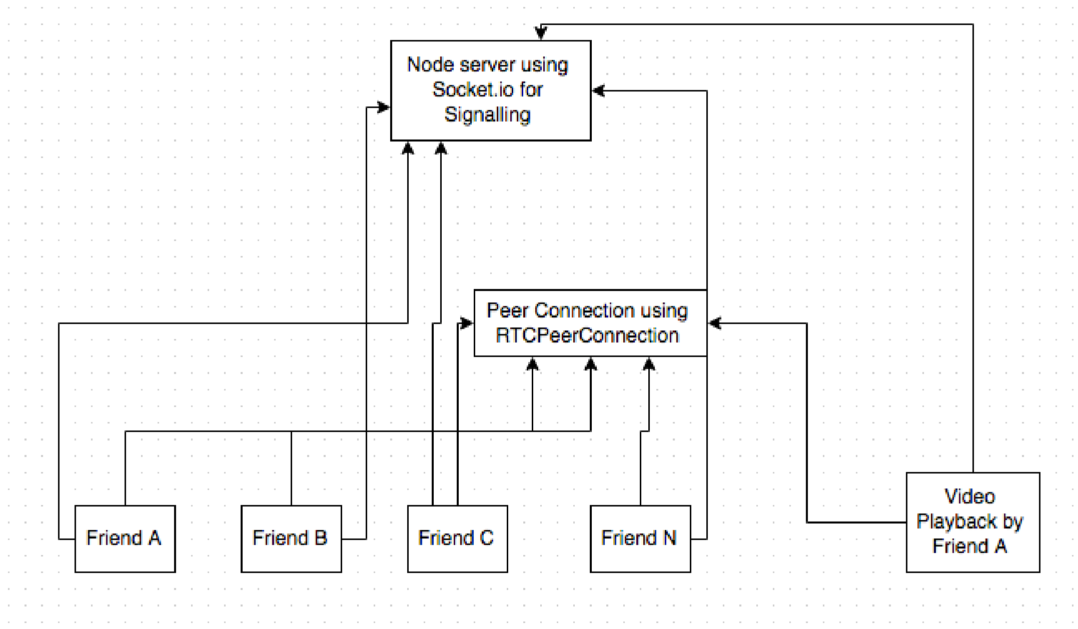
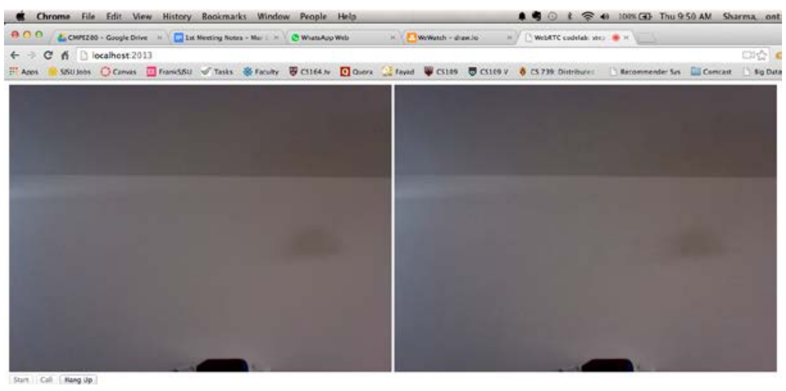

Node server using Socket.io for Signalling
Peer Connection using RTCPeerConnection
Friend A
Friend B
Friend C
Friend N
Video Playback by Friend A

FIGURE 18. WEWATCH ARCHITECTURE

FIGURE 19. WEWATCH SCREEN CAPTURE



D. Connection Clinic Prototype

I. Purpose

Connection Clinic is a medical app that 'connects' doctors, nurses, and patients in an app to easily transmit information. For example, a doctor may want to send or retrieve information from a nurse to diagnose and review information with a patient in real-time. Live chat support is a new tool with growing support, but the proposed prototype is different. The idea and method has some similarities, which is live support and connection to the respective professional according to the customer's needs. The disparity rises in the sophistication of the interaction. Instead of a chat client, the user can do the following: (1) chat and share notes with a doctor or nurse, (2) video chat, and (3) see doctor's notes and recommendation.

The app will have different views depending on the user. The doctor may need to look at the patient history or may want to have confidential side conversations with the staff, whereas the patient may want to take notes or save information from a consulting session. WebRTC is an ideal technology for this prototype because of the bi-directional real-time exchange of data across platforms and devices will facilitate a consolidated straightforward method for interacting with patients and medical professionals.

To successfully implement a scalable, user friendly, and efficient medical chat system, several sections are needed: (1) webRTC to connect all users (i.e. doctors, nurses, patient), (2) an effective UI, and (3) an architecture that is reliable and accounts for the varying permissions for each end user. To create a solution for the medical industry, the system architecture, appropriate wireframes, applied web UI design principles, sequence diagrams, and anticipated results shall be discussed.

II. Architecture

Below is the proposed architecture for the system. The idea is to use TURN to optimize the system.

For traffic management, there are several options, such as node.js, which is a light weight, even driven, asynchronous framework that efficiently can manage requests. Additional details shall be included in future versions.

III. Wireframes

There will be additional pages, such as login. The landing pages are presented below. To uphold UI design principles of consistency and familiarity, we have constructed similar pages for the patient and doctor. At a high level view, both pages seem the same, but the difference resides within the options. For example, a user has the ability to 'Ask a Question' or 'Request a Person' under the Chat dropdown. The doctor has the same format, but altered options that are 'Request a Patient,' 'Request a medical professional,' or view a chat screen. The doctor may have a separate chat screen with a specialist that he or she may not want viewed by the patient. This idea is a work in progress and shall be expanded in future work.

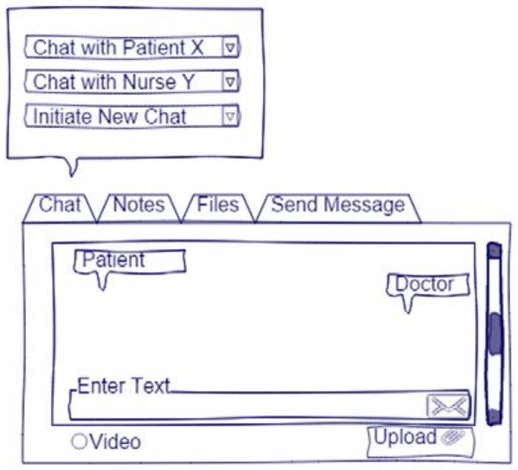


FIGURE 21. CONNECTION CLINIC WIREFRAME

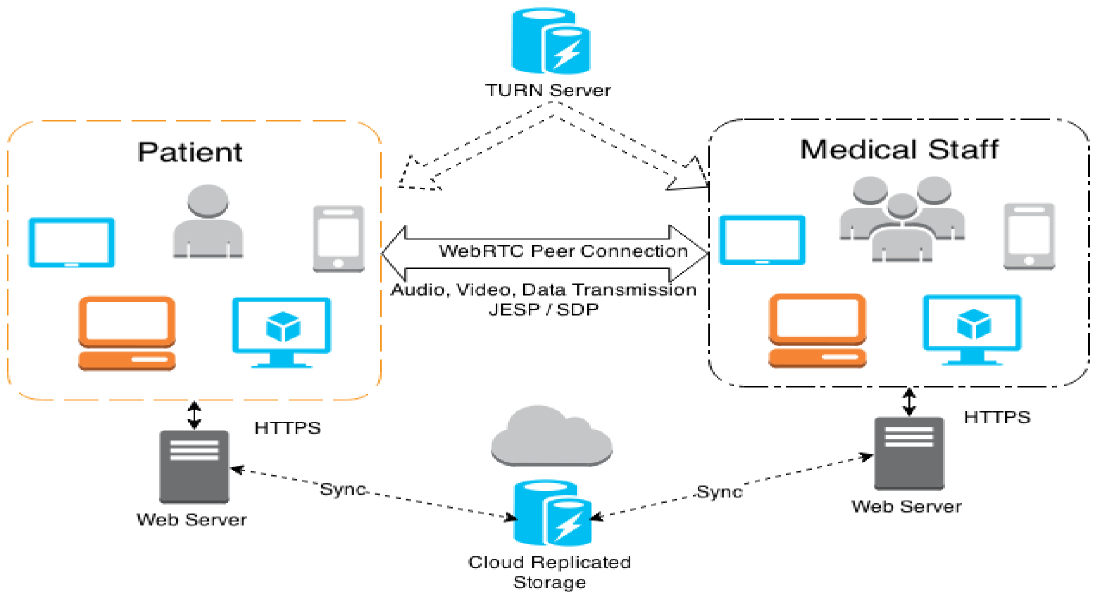


FIGURE 20 CONNECTION CLINIC ARCHITECTURE

III. Wireframes

When crafting the wireframes, the following UI design principles were incorporated. Below shows a principle, brief description, and how the principle is applied in the Connection Clinic system.

FIGURE 22. USER DESIGN PRINCIPLES IMPLEMENTED IN PROTOTYPE [13]

Principle	Description	Application
Accessibility	System is usable without alterations and has “perceptibility, operability, simplicity, and forgiveness” characteristics.	System for all users has the same look and feel so doctors, nurses, and patients can easily find and guide people in the application.
Aesthetically Pleasing	System uses graphics and other visual elements to the user’s liking.	The application will have a simple and straightforward design so a user can easily find and send information.
Availability and Responsiveness	Do not limit the user’s interaction with the system.	All options will be available upon sign in and the users can send feedback.
Clarity and Visibility	All operations and visual elements should be clear.	This will be accomplished with a simple design that effectively combines white space and graphics.
Compatibility	Create a system that accommodates the user’s needs.	The system can be used on mobile and internet devices. Future versions may include the television.
Configurability	Allow the user to modify and be an active participant in the system.	This is accomplished in the chat interface and when users send and retrieve files.
Consistency	The system should have the same usability and feel for the entire application.	The patient, doctor, and other medical professional landing pages are all the same except for the options in the dropdown and page heading.
Familiarity, Predictability, and Obviousness	Use logical and customary functions for the system flow.	A review shall be conducted to analyze commonalities of medical communication systems to determine an effective mechanism that fulfills this UI design principle.
Forgiveness	The system should allow users to easily remedy errors or a decision change.	The idea is to have everything on the main page with tabs so that the need to go ‘back’ or remedy errors will be lessened. All standard options like ‘cancel upload’ and others will be included.

BEST PRACTICES AND CODING STANDARDS

WebRTC application is most prevalently embedded in a web page hosted by an enterprise. Below are some of the key points which will provide good user experience while designing a webpage supporting WebRTC application:

- Chat/call window should not be tightly coupled with the functioning of web page. The user should not be bound to WebRTC window and best possible effort should be given on enforcing the user to use the webpage independently apart from initiating a call/chat/video conversation.
- Positioning of the video or chat window should be done carefully. The conversation should not overlap any of the important website features.
- Chat should be rendered in such way such that the users can send/receive any links/multimedia files over the chat to make troubleshooting easier.
- Full-screen video would enhance the overall user experience

CONCLUDING REMARKS

WebRTC, an open source project, is a powerful and simple web technology for real-time communications. In this paper we have introduced WebRTC core concepts, discussed few case studies and proposed our web application ideas, WeWatch and Talk2Doc. Real-time communications is been here for decades. To understand the importance of WebRTC, we have made an analysis of its existing peers like PSTN, SIP and Flash as part of our survey. Then we have explained WebRTC concepts, architecture, explained its core components: WebRTC C++ API and WebRTC API in detail with code snippets and examples. We have demonstrated steps to establish a simple connection between a caller and callee using WebRTC API's RTCPeerConneciton. WebRTC by itself fail to handle remote server connections and indicate the need for signaling and media communication mechanisms. Signaling for meta data exchange and data exchange is done using various external technologies like Websockets, SDP, JSEP, SIP etc. We have demonstrated steps for establishing connection between two remote servers using Session Description Protocol. Also the media communication technologies like STUN/TURN servers, ICE frameworks are have been addressed. Other aspects of WebRTC such as security, its cost and application deployment are addressed as well.

In the later part of the paper we have presented two case studies and a prototype of our idea, 'WeWatch'. Case studies, 'A Video Conferencing System Based on WebRTC for Seniors' and 'An On-Demand WebRTC and IoT Device Tunneling Service for Hospitals' are elaborated to better understand the real-world web application using WebRTC. The purpose, overview, technology stack of case studies are presented. Finally, we presented our project idea WeWatch, where friends can watch a video together virtually. Here we have given basic architecture and the output of our prototype.

All in all, WebRTC is a web technology, which opens up new possibilities. It has the power to turn simple web applications to communication tools. Let's all accept that this technology is for here to stay in the coming decades.

REFERENCES

[1] Audin, G. (2014, 01 03). *9 Advantages of WebRTC*. Retrieved from Information Week: Network Computing, Connecting the Infrastructure Community: http://www.networkcomputing.com/unified-communications/9-advantages-of-webrtc/a/d-id/1113301

[2] Barnes, R., & Thomson, M. (2014). Browser-to-Browser Security Assurances for WebRTC. *Internet Computing, IEEE, 18*(6), 11 - 17. doi:10.1109/MIC.2014.106

[3] Boccamazzo, A. (2014, August 13). *Six Key Benefits of WebRTC That You Need to Know*. (WebRTC wORLD) Retrieved March 28, 2015, from http://www.webrtcworld.com/topics/webrtc-world/articles/386574-six-key-benefits-webrtc-that-need-know.htm

[4] Elleuch, W. (2013). Models for multimedia conference between browsers based on WebRTC. *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, 279 - 284. doi:10.1109/WiMOB.2013.6673373

[5] Galitz, W. (2007). *The Essential Guide to User Interface Design*. Indianapolis: Wiley Publishing.

[6] Google Developers. (2013). Real-time communicatio with WebRTC: Google I/O 2013. Retrieved from https://www.youtube.com/watch?v=p2HzZkd2A40

[7] Groenfeldt, T. (2015, March 24). *Future Crimes: Tech Threats From Hackers, China, Google And Facebook*. (Forbes) Retrieved March 2015, 25, from http://inhomelandsecurity.com/future-crimes-tech-threats-from-hackers-china-google-and-facebook/

[8] Oracle. (2015). *Overview of Cross-Platform Issues*. (Oracle Fusion Middleware Online Documentation Library) Retrieved March 26, 2015, from http://docs.oracle.com/cd/E14571_01/bi.1111/b32121/pbr_xplat003.htm#RSPUB23673

[9] PRWeb. (2015, March 31). *Error Reporting Software Provider Raygun, Surveyed 1,400 Developers About How They Handle Software Errors*. (Benzinga) Retrieved March 2015, 31, from http://www.benzinga.com/pressreleases/15/03/p5373931/error-reporting-software-provider-raygun-surveyed-1-400-developers-abou

[10] Risen, T. (2014, June 9). *Study: Hackers Cost More Than $445 Billion Annually*. (US News and World Report) Retrieved March 28, 2015, from http://www.usnews.com/news/articles/2014/06/09/study-hackers-cost-more-than-445-billion-annually

[11] SightCall. (2014). *Five Reasons to Choose WebRTC for Video Calling*. (SightCall) Retrieved March 2015, 28, from http://www.sightcall.com/five-reasons-choose-webrtc-video-calling/

[12] Warren, C. (2012, February 16). *The Pros and Cons of Cross-Platform App Design*. (Mashable) Retrieved March 25, 2015, from http://mashable.com/2012/02/16/cross-platform-app-design-pros-cons/

[13] Dutton, S. (n.d.). *simpl*. Retrieved March 29, 2015, from simpl.info: http://simpl.info/

[14] Eric, R. (n.d.). *WebRTC Security Architecture*. Retrieved Mar 31, 2015, from webrtc: http://www.webrtc.org/architecture