Web Application for Social Networking using RTC

Nileshkumar Pandey
Department of Computer Science
California State University, Fullerton
Fullerton, CA, USA
nileshpandey3@csu.fullerton.edu

Doina Bein
Department of Computer Science
California State University, Fullerton
Fullerton, CA, USA
dbein@fullerton.edu

***Abstract*— Real-time communication (RTC) is an important part of modern convergence networks [1]. Interactive voice communication, video, conferencing, chat, messaging and presence are today very popular and widely use IP network services and applications. Many web services already use RTC, but need downloads, native apps or plugins. These includes Skype, Facebook (which uses Skype) and Google Hangouts (which use the Google Talk plugin). Downloading, installing and updating plugins can be complex, error prone and annoying. Plugins can be difficult to deploy, debug, troubleshoot, test and maintain—and may require licensing and integration with complex, expensive technology. The vision of WebRTC is a world where smartphones, TVs and computers could all communicate on a common platform. The guiding principles of the WebRTC project are that its APIs should be open source, free, standardized, built into web browsers and more efficient than existing technologies. WebRTC has implemented open standards for real-time, plugin-free video, audio and data communication. The objective of this paper is to present a web application for networking and interaction between students and instructs using the concept of real-time peer-to-peer communication (WebRTC). The web application serves as a Q&A forum as well as a networking site for students to interact with the instructor as well as peer students by posting their questions in Twitter-style tweets and have other students or the instructor answer them [2].**

Keywords— Online forum for supporting learning; plugin free communication; real-time communication; web application.

I. INTRODUCTION

One of the last major challenges for the web is to enable human communication via voice and video: Real Time Communication, RTC for short. RTC should be as natural in a web application as entering text in a text input. Without it, we are limited in our ability to innovate and develop new ways for people to interact. Historically, Real Time Communication has been corporate and complex, requiring expensive audio and video technologies to be licensed or developed in house. Integrating RTC technology with existing content, data and services has been difficult and time consuming, particularly on the web. The WebRTC technology [1] natively integrates multimedia communication capabilities into modern web browsers, and enables them to communicate on a peer-to-peer basis using audio, video and data, without a need for special purpose client, neither external plug-ins, nor extensions [2]. The design evolution, standardization and technology integration of WebRTC are subjects of several standardization bodies, mainly the World Wide Web Consortium (W3C) and the Internet Engineering Task Forces (IETF). WebRTC work is focusing mainly on peer-to-peer browser based multimedia communication, working on the vision of providing universal, open and browser-based communication framework without a need for special purpose communication end-devices [3]. The idea of WebRTC is simple and revolutionary, leading the communication evolution forward to the browser based, and special-purpose clientless future [4].

This paper presents [5] a web application for networking and interaction between students and instructs using the concept of real-time peer-to-peer communication (WebRTC). The web application will serve as a Q&A forum as well as a networking site for students to interact with the instructor as well as peer students by posting their question or queries in Twitter-style tweets and have other students or the instructor answer them.

The paper is organized as follows. In Section II we present the basic notions regarding real-time communication, integrating real-time communication media into a web page, and WebRTC API. A description of the project functionality is given in Section III. Concluding remarks and future work are presented in Section IV.

II. REAL-TIME COMMUNICATION ON THE WEB

Real-time communication (RTC) is an important part of modern convergence networks. Interactive voice communication, video, conferencing, chat, messaging and presence are today very popular and widely used IP network services and applications. Many web services already use RTC, but need downloads, native apps or plugins. These includes Skype, Facebook (which uses Skype) and Google Hangouts (which use the Google Talk plugin). Downloading, installing and updating plugins can be complex, error prone and annoying. Plugins can be difficult to deploy, debug, troubleshoot, test and maintain—and may require licensing and integration with complex, expensive technology.

The main idea behind the WebRTC development was a need to offer a native web browser based tool that uses web technologies, to provide a simple way to insert and use real –

[1] This paper is based on the Master project report of Nileshkumar Pandey, submitted to the Department of Computer Science at California State University, Fullerton, on May 2017.

[2] Doina Bein acknowledges the support by Air Force Office of Scientific Research under award number FA9550-16-1-0257.

978-1-5386-4649-6/18/$31.00 ©2018 IEEE

time based communication media (as audio, video, data transfer), directly into web page [6]. Then the knowledge of usual web technologies (HTML and JavaScript) will be enough to integrate a real-time communication into a web page. The vision of WebRTC is a world where smartphones, TVs and computers could all communicate on a common platform. The WebRTC development effort was driven by Google as an open source project, but the project is also supported by other web companies (Mozilla, Opera) or telecom equipment vendors (Ericsson, Cisco, Alcatel/Lucent). WebRTC enables web browsers with Real-Time Communications (RTC) capabilities natively, without the need for installing additional third-party software or plugins. WebRTC has implemented open standards for real-time, plugin-free video, audio and data communication. The guiding principles of the WebRTC project are that its APIs should be open source, free, standardized, built into web browsers and more efficient than existing technologies.

WebRTC offers web application developers the ability to write rich, real-time multimedia applications (think video chat) on the web, without requiring plugins, downloads or installs. Its purpose is to help build a strong RTC platform that works across multiple web browsers, across multiple platforms. There are two distinct layers:

1. For Browser development, we have WebRTC C++ API and the capture/render hooks at their disposal. WebRTC API is an API layer that enables browser makers to easily implement the Web API proposal.
2. For Web App development, we have the Web API, an API to be used by third party developers for developing web based video chat-like applications.

There are several WebRTC standardization activities under the lead of W3C and IETF. Standardization separates the roles and responsibilities between the single node (application level responsibilities) and internode communication (media and signaling exchange) [6]. Their joined efforts are focusing on defining both the HTML JavaScript APIs (W3C), and the underlying networking communication protocols for the setup management of reliable peer-to-peer multimedia communications between pairs of browsers (IETF). The WebRTC API provides means for creating browser based client-side RTC applications with functions like browser-to-browser connection management, negotiation, media control, encoding/decoding, NAT traversal and etc..

WebRTC implements three APIs, MediaStream, RTCPeerConnection and RTCDataChannel API. They are incorporated within an overall WebRTC architecture. The Media Stream API allows web browser applications access to the media streams from the user's camera and microphone. RTCPeerConnection allows the audio or video session establishment (offer/accept negotiation), with facilities for encryption and bandwidth management. The RTCPeerConnection API does not provide, but require, a connection control mechanism, aka signaling. The WebRTC developers may reuse the functionality of already developed internet session control protocols like SIP [7] and XMPP (Extensible Messaging and Presence Protocol). The RTCDataChannel API provides functionality for peer-to-peer data communication exchange over a bi-directional data channel. RTCDataChannel API natively support the establishment of several and secured communication channels.

Enabling a rich teleconferencing experience in the browser requires access to the system hardware in order to capture both audio and video—no third-party plug-ins or custom drivers, just a simple and a consistent API [8]. However, raw audio and video streams are also not sufficient on their own: each stream must be processed to enhance quality, synchronized, and the output bitrate must adjust to the continuously fluctuating bandwidth and latency between the clients. On the receiving end, the process is reversed, and the client must decode the streams in real-time and be able to adjust to network jitter and latency delays. In short, capturing and processing audio and video is a complex problem. However, WebRTC brings fully featured audio and video engines to the browser, which take care of all the signal processing, and more. The acquired audio stream is processed for noise reduction and echo cancellation, then automatically encoded with one of the optimized narrowband or wideband audio codecs. Finally, a special error-concealment algorithm is used to hide the negative effects of network jitter and packet loss—that's just the highlights! The video engine performs similar processing by optimizing image quality, picking the optimal compression and codec settings, applying jitter and packet-loss concealment, and more.

All of the processing is done directly by the browser, and even more importantly, the browser dynamically adjusts its processing pipeline to account for the continuously changing parameters of the audio and video streams and networking conditions. Once all of this work is done, the web application receives the optimized media stream, which it can then output to the local screen and speakers, forward to its peers, or post-process using one of the HTML5 media APIs.

The Media Capture and Streams W3C specification defines a set of new JavaScript APIs that enable the application to request audio and video streams from the platform, as well as a set of APIs to manipulate and process the acquired media streams. The MediaStream object is the primary interface that enables all of this functionality and consists of one or more individual tracks (MediaStreamTrack). Tracks within a MediaStream object are synchronized with one another. The input source can be a physical device, such as a microphone, webcam or a local or remote file from the user's hard drive or a remote network peer. The output of a MediaStream can be sent to one or more destinations: a local video or audio element, JavaScript code for post-processing, or a remote peer. A MediaStream object represents a real-time media stream and allows the application code to acquire data, manipulate individual tracks, and specify outputs. All the audio and video processing, such as noise cancellation, equalization, image enhancement, and more are automatically handled by the audio and video engines.

getUserMedia() API is responsible for requesting access to the microphone and camera from the user, and acquiring the streams that match the specified constraints—that's the whirlwind tour. The provided APIs also enable the application to manipulate individual tracks, clone them, modify

constraints, and more. Further, once the stream is acquired, we can feed it into a variety of other browser APIs. Web Audio API enables processing of audio in the browser. Canvas API enables capture and post-processing of individual video frames. CSS3 and WebGL APIs can apply a variety of 2D/3D effects on the output stream. getUserMedia() is a simple API to acquire audio and video streams from the underlying platform. The media is automatically optimized, encoded, and decoded by the WebRTC audio and video engines and is then routed to one or more outputs. With that, we are halfway to building a real-time teleconferencing application—we just need to route the data to a peer.

Real-time communication is time-sensitive; that should come as no surprise. As a result, audio and video streaming applications are designed to tolerate intermittent packet loss: the audio and video codecs can fill in small data gaps, often with minimal impact on the output quality. Similarly, applications must implement their own logic to recover from lost or delayed packets carrying other types of application data. Timeliness and low latency can be more important than reliability. The requirement for timeliness over reliability is the primary reason why the UDP protocol is a preferred transport for delivery of real-time data. TCP delivers a reliable, ordered stream of data: if an intermediate packet is lost, then TCP buffers all the packets after it, waits for a retransmission, and then delivers the stream in order to the application. By comparison, UDP offers no promises on reliability or order of the data, and delivers each packet to the application the moment it arrives. In effect, it is a thin wrapper around the best-effort delivery model offered by the IP layer of our network stacks.

WebRTC uses UDP at the transport layer: latency and timeliness are critical. With that, we can just fire off our audio, video, and application UDP packets, and we are good to go, right? Well, not quite. We also need mechanisms to traverse the many layers of NATs and firewalls, negotiate the parameters for each stream, provide encryption of user data, implement congestion and flow control, and more. UDP is the foundation for real-time communication in the browser, but to meet all the requirements of WebRTC, the browser also needs a large supporting cast of protocols and services above it.

Despite the many protocols involved in setting up and maintaining a peer-to-peer connection, the application API exposed by the browser is relatively simple. The RTCPeerConnectioninterface is responsible for managing the full life cycle of each peer-to-peer connection. RTCPeerConnection encapsulates all the connection setup, management, and state within a single interface.

Initiating a peer-to-peer connection requires (much) more work than opening an XHR, EventSource, or a new WebSocket session: the latter three rely on a well-defined HTTP handshake mechanism to negotiate the parameters of the connection, and all three implicitly assume that the destination server is reachable by the client—i.e., the server has a publicly routable IP address or the client and server are located on the same internal network.

Before any connectivity checks or session negotiation can occur, we must find out if the other peer is reachable and if it is willing to establish the connection. We must extend an offer, and the peer must return an answer (Figure 6). However, now we have a dilemma: if the other peer is not listening for incoming packets, how do we notify it of our intent? At a minimum, we need a shared signaling channel.

WebRTC defers the choice of signaling transport and protocol to the application; the standard intentionally does not provide any recommendations or implementation for the signaling stack. Why? This allows interoperability with a variety of other signaling protocols powering existing communications infrastructure, such as the following:

- Session Initiation Protocol (SIP): an application-level signaling protocol, widely used for voice over IP (VoIP) and videoconferencing over IP networks.
- Jingle: signaling extension for the XMPP protocol, used for session control of voice over IP and videoconferencing over IP networks.
- ISDN User Part (ISUP): signaling protocol used for setup of telephone calls in many public switched telephone networks around the globe. A WebRTC application can choose to use any of the existing signaling protocols and gateways to negotiate a call or a video conference with an existing communication system—e.g., initiate a "telephone" call with a PSTN client! Alternatively, it can choose to implement its own signaling service with a custom protocol.
- Audio, Video, and Data Streaming: Peer-to-peer audio and video streaming are one of the central use cases for WebRTC: getUserMedia API enables the application to acquire the media streams, and the built-in audio and video engines handle the optimization, error recovery, and synchronization between streams. However, it is important to keep in mind that even with aggressive optimization and compression, audio and video delivery are still likely to be constrained by latency and bandwidth: An HD quality streams requires 1–2 Mbps of bandwidth. The global average bandwidth as of Q1 2013 is just 3.1 Mbps. An HD stream requires, at a minimum, a 3.5G+ connection.

III. PROJECT DESCRIPTION

In this section, we present the modules on the project. Currently this technology is supported in only in Chrome/Mozilla Firefox and MS Edge browser, but Chrome is recommended. Hardware requirements: processor Intel core i3 or higher, 160 GB hard disk or higher, 4 GB RAM or higher. Software requirements: development languages (HTML, CSS, JavaScript, Rails 5, node JS) and Linux or MacOS as operating systems.

The main website is presented in Fig. 1. To access the web application, a user needs to sign-up (see Fig. 2). After signing-up, the user can sign-in (Fig. 3). User can post new tweets with text and upload images if needed (Fig. 4).

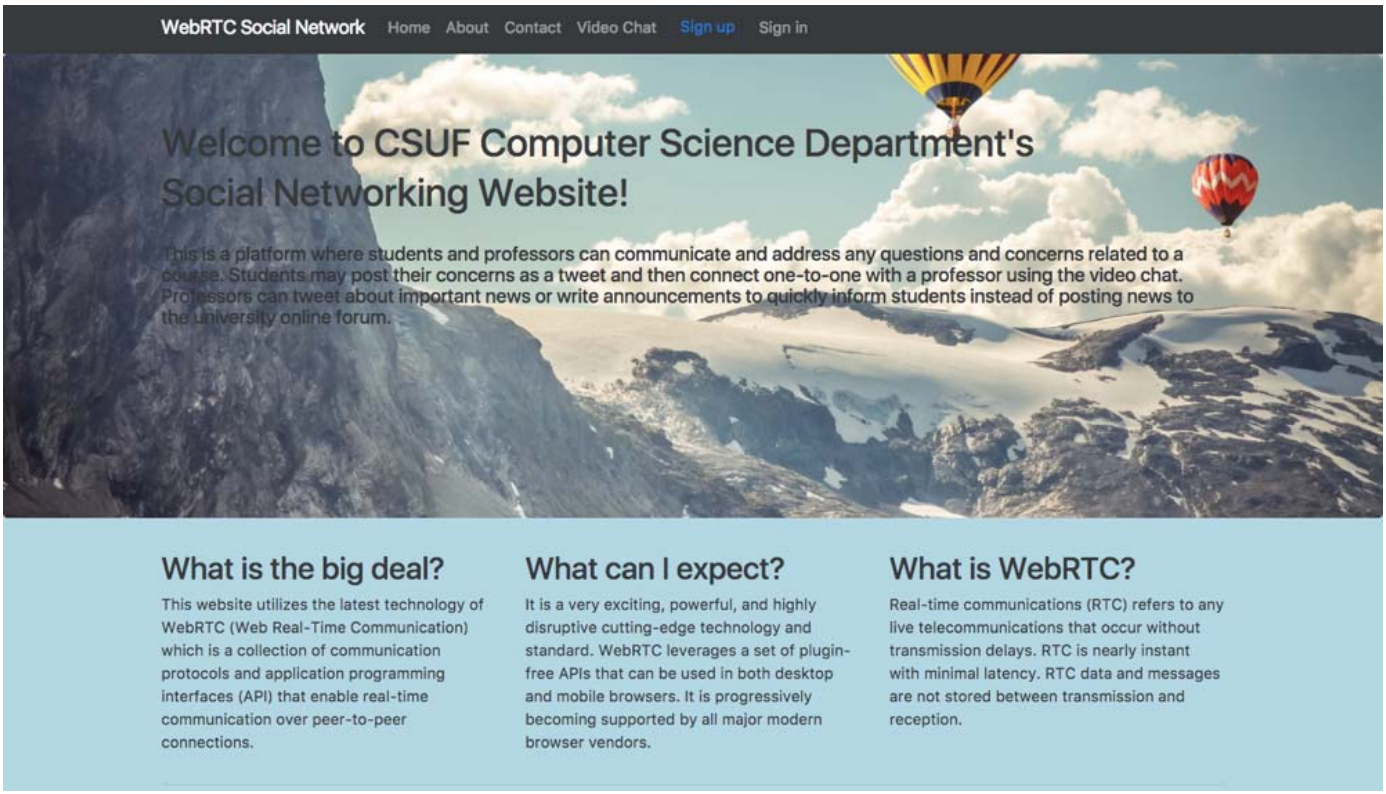

Figure 1. Main website

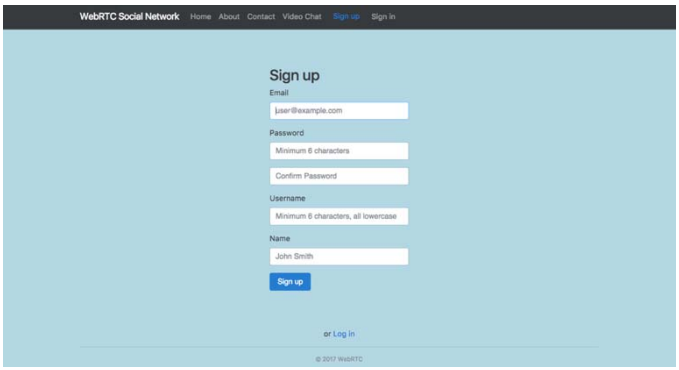

Figure 2. New user sign-up

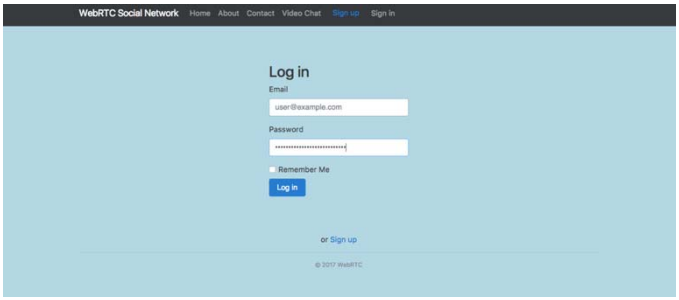

Figure 3. Existing user signing in

User can follow other users and the tweets posted by other users will be displayed ion the user's wall feed page (Fig. 5). Finally, the user can video chat in real time with instant messaging option (see Fig. 6). An example is shown in Fig. 7.

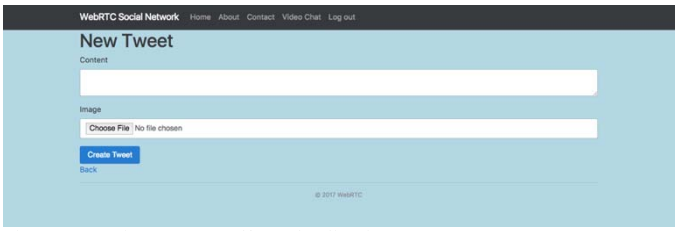

Figure 4. Posting tweets and/or uploading images

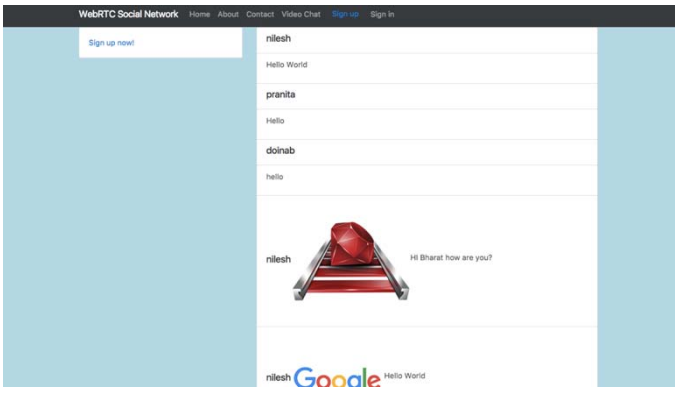

Figure 5. Viewing tweets from friends

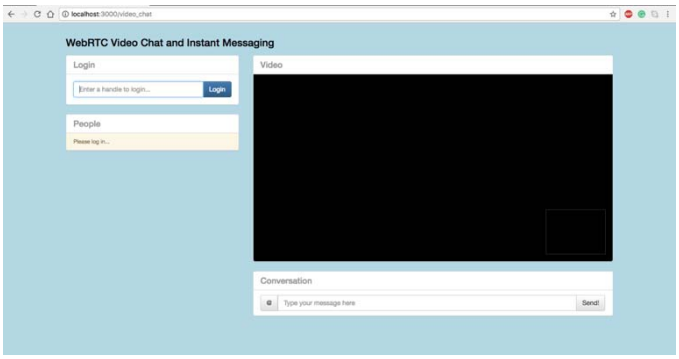

Figure 6. Video chat functionality

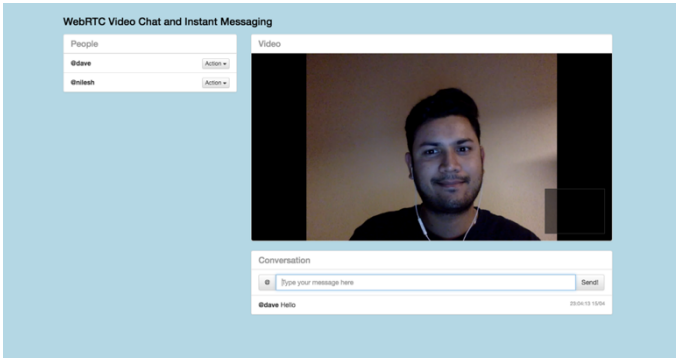

Figure 7. User can video chat in real time with instant messaging option

IV. Conclusion and Future Work

The objective of this paper is to present a web application for networking and interaction between students and instructs using the concept of real-time peer-to-peer communication (WebRTC). The web application serves as a Q&A forum as well as a networking site for students to interact with the instructor as well as peer students by posting their questions in Twitter-style tweets and have other students or the instructor answer them. The current system performs only minimal functionality. This project can be extended to include real time data transfer e.g. files and document during the video/audio chat feature. The system can also be extended to encrypt the video/audio/text communication channel to secure the communication between two or multiple parties.

References

[1] WebRTC, "WebRTC," 3 May 2011. [Online]. Available: https://webrtc.org/. [Accessed 19 November 2017].

[2] S. P. Romano and S. Loreto, Real-Time Communication with WebRTC, O'Reilly Media, Inc., 2014.

[3] I. Grigorik, High Performance Browser Networking, O'Reilly, 2013.

[4] R. Vapenik, M. Michalko, J. Janitor and F. Jakab, "Secured web oriented videoconferencing system for educational purposes using WebRTC technology," in *IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2014.

[5] N. Pandey, "Real Time Peer to Peer Communication based Social Networking Web Application," Department of Computer Science, California State University, Fullerton, Fullerton, 2017.

[6] S. Dutton, 4 November 2013. [Online]. Available: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/. [Accessed 19 November 2017].

[7] P. Segec, P. Paluch, J. Papan and M. Kubina, "The integration of WebRTC and SIP: Way of enhancing real-time, interactive multimedia communication," in *IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2014.

[8] K. Singh and J. Buford, "Developing WebRTC-based team apps with a cross-platform mobile framework," in *13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2016.