WebRTC technology overview and signaling solution design and implementation

Branislav Sredojev*, Dragan Samardzija ** and Dragan Posarac*
* Faculty of Technical Sciences, University of Novi Sad, 21000 Novi Sad, Republic of Serbia
** RT-RK Institute for Computer Based Systems, 21000 Novi Sad, Republic of Serbia
sredojev.branislav@gmail.com

Abstract - This paper describes the WebRTC technology and implementation of WebRTC client, server and signaling. Main parts of the WebRTC API are described and explained. Signaling methods and protocols are not specified by the WebRTC standards, therefore in this study we design and implement a novel signaling mechanism. The corresponding message sequence chart of the WebRTC communication behavior describes a communication flow between peers and the server. The server application is implemented as a WebSocket server. The client application demonstrates the use of the WebRTC API for achieving real-time communication. Benefits and future development of the WebRTC technology are mentioned.

I. INTRODUCTION

WebRTC (Web Real-Time Communication) is a new web technology which allows browser and mobile applications with functionalities such as audio/video calling, chat, P2P (peer-to-peer) file sharing and all that without any additional third-party software or plugins.

It was published as an open source technology by Google in May 2011 and includes fundamental components for real-time communication on the web. With those components, that can be accessed via the JavaScript API (Application Programming Interface), developers have opportunity to create impressive web applications. Google, Mozilla and Opera support WebRTC and they will participate in the further development of this technology. At the moment, there are still platforms and browsers that either partially or do not support WebRTC [1]. Major components of the WebRTC API are:

- MediaStream - allows a web browser to access the camera and microphone;
- RTCPeerConnection - sets up audio or video calls;
- RTCDataChannel - allows browsers to send data through peer-to-peer connections.

It is important to know that WebRTC is not just a single API, but it is a collection of APIs and protocols defined by the various working groups such as W3C (World Wide Web Consortium) and IETF (Internet Engineering Task Force). Support for each of them develops in different browsers and operating systems [2].

A. MediaStream

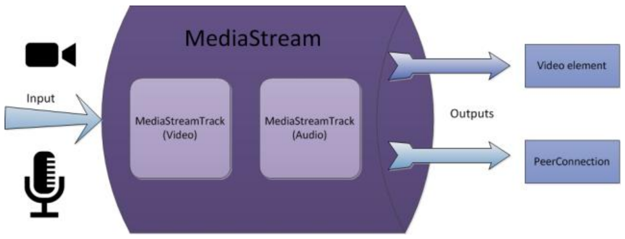


Figure 1. MediaStream

The MediaStream API has a control over the synchronized streams of media [3]. Each MediaStream has an input and output. The input source might be a physical device, such as the camera or microphone. The output might be forwarded to one or more destinations, such as video element or PeerConnection (Figure 1.).

The two main components in the MediaStream API are the MediaStream interface and MediaStreamTrack. The MediaStream interface represents a stream of media content. That media stream can have several parts. Those parts are called tracks and they have explicit type, such as audio and video. MediaStreamTrack represents a type of media that have been obtained from the input source. For example, if it is a microphone, we will get MediaStreamTrack (Audio).

B. RTCPeerConnection

The RTCPeerConnection API allows two peers to communicate directly, browser to browser. The WebRTC architecture diagram (Figure 2.) represents a very simple hierarchy. WebRTC provides a framework to make real-time communications possible.

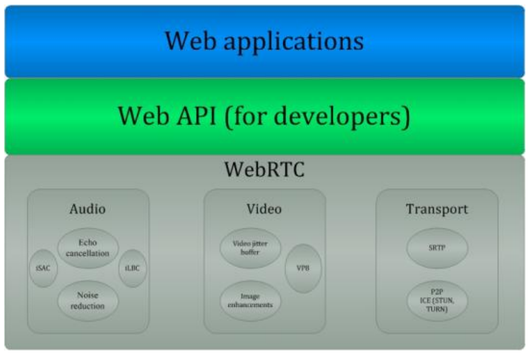


Figure 2. WebRTC architecture

The WebRTC architecture has three basic sections. The first section, WebRTC, consists of an audio, video and transport mechanism.

The main task of the audio mechanism is to handle echo cancellation and noise reduction. Echo cancellation is a method that improves the sound quality, removing the resulting echo or preventing its occurrence. Noise reduction is the process of removing noise from the signal. The audio mechanism contains iSAC and iLBC codecs. iSAC codec represents a wideband audio codec developed by Global IP Solutions. This codec is suitable for audio streams and in June 2011 became a part of the WebRTC technology. Global IP Solutions is also the creator of the iLBC codec. This narrowband audio codec is suitable for voice communication over IP.

The video mechanism is responsible for image enhancement and video jitter buffer. Image enhancement is the process of improving the quality of digital image by manipulation with a certain software. The video mechanism includes VP8 codec. VP8 codec is a video compression format. It was developed by On2 Technologies in September 2008 and is currently supported by Chrome, Mozilla and Opera.

The transport mechanism is responsible for SRTP (Secure Real-time Transport Protocol), P2P and ICE (Interactive Connectivity Establishment). P2P and ICE with STUN and TURN servers will be explained in the following chapter.

The Web API is on the top of the WebRTC section which shields developers from complexities and facilitates their work. The third and last section are Web applications, such as audio, video and chat applications with possibility of file sharing.

C. *RTCDataChannel*

The RTCDataChannel API represents a bi-directional data channel between two peers and enables exchange of arbitrary data between them. Each RTCDataChannel can be configured to provide:

- reliable or unreliable delivery of messages;
- in-order or out-of-order delivery of messages.

The reliable and in-order delivery is equivalent to TCP (Transmission Control Protocol). On the other hand, the unreliable and out-of-order delivery is equivalent to UDP (User Datagram Protocol). There are many potential use cases for the API, including:

- gaming;
- remote desktop applications;
- real-time text chat;
- file transfer.

D. *ICE*

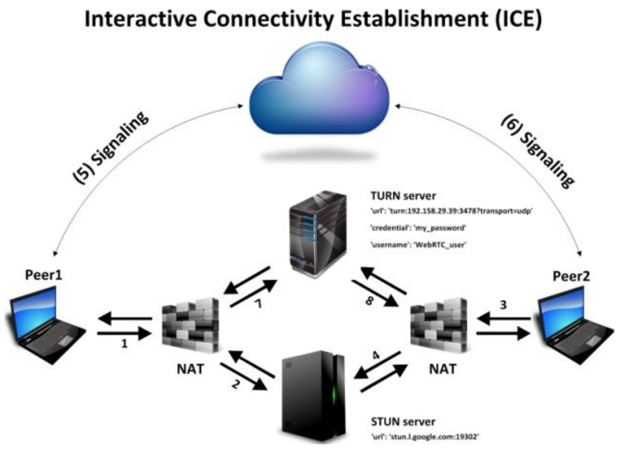

Interactive Connectivity Establishment (ICE)
(5) Signaling
(6) Signaling
TURN server
'url': 'turn:192.158.29.39:3478?transport=udp'
'credential': 'my_password'
'username': 'WebRTC_user'
Peer1
Peer2
1
2
3
4
7
8
NAT
NAT
STUN server
'url': 'stun.l.google.com:19302'

Figure 3. Interactive Connectivity Establishment (ICE)

The Internet addressing system is still using IPv4 (Internet Protocol version 4) and because of that, most of our devices are behind one or more layers of NAT (Network Address Translation). NAT is a mechanism of mapping private addresses to the public address changing address information within IP packets while it is in transit through the device for routing. NAT can be implemented with a home Wi-Fi router.

WebRTC technology developers can use ICE which will simplify complexity of the Internet addressing system. Developers need to pass the ICE server URLs to the peer connection. ICE will try to find the best path to connect two peers (Figure 3.). ICE is using two types of servers, STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT). ICE is using a STUN server to find out what external address is assigned to a specific peer. If that fails, the traffic is routed via a TURN server. Every TURN server supports STUN, a TURN server is a STUN server with added relaying functionality built in [4].

An analysis by webrtcstats.com in 2014, which lasted 6 months covering over 6 million minutes of one-to-one video calls, shows that 92% of WebRTC video calls are P2P without a relay and just 8% are over a TURN server (Figure 4.) [5].

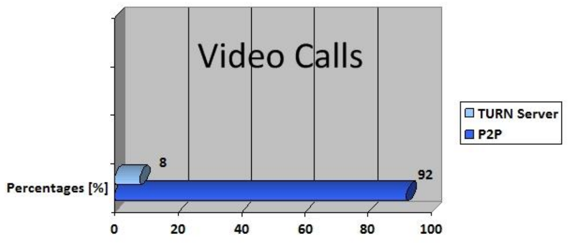


Figure 4. Video calls statistics

II. DESCRIPTION OF REALIZATION

In this section, we describe our implementation of a WebRTC client, server and newly-proposed signaling solution between them. Signaling is a process of exchanging system control information like session control messages, media metadata, error messages and network data - information important to set up a call. This signaling process needs a two-way (bi-directional) messaging and therefore we have chosen to implement signaling using WebSocket. The server application is implemented as a WebSocket server. The client application demonstrates the use of the WebRTC API for achieving real-time communication.

A. Signaling

Signaling methods are not specified by WebRTC in order to avoid redundancy and to maximize compatibility with established technologies [6].

The signaling process corresponds to protocols such as SIP (Session Initiation Protocol), XMPP (Extensible Messaging and Presence Protocol), XHR (Xml Http Request), and SIP over WebSocket. In this study, the signaling implementation uses the WebSocket mechanism [7]. WebSocket is a relatively new addition to browsers, and it was standardized in 2011. The most important thing about WebSocket is that it opens a session from a peer to the server, leaving it open until the session is closed. Messages can be textual or binary. The message sequence chart of the WebRTC communication describes communication flow between the peers and server. The diagram is depicted in Figure 6.

In our signaling solution we made four types of the control messages: “initialize”, “initiator”, “got user media” and “peerChannel”. Those control messages are textual and they are used to set up flags for certain mechanisms. At start, Peer1 will send “initialize” to the server, because it needs to find out whether it is the session initiator. If so, it will configure the user media after getting the control message “got user media” from the server. Now Peer1 is in a waiting phase until Peer2 appears. Upon its activation, Peer2 sends the equivalent signaling message, and when it finds out that it isn’t the initiator, it will first get “peerChannel” and then “got user media”. After that, both sides can start establishing a peer-to-peer connection.

The offer and answer messages are transmitted in the SDP (Session Description Protocol) format [8]. SDP plays a main role in the setup of WebRTC sessions. SDP is the protocol that has been chosen by IETF and W3C to exchange media information between peers in WebRTC. SDP carries information about media type, format, codecs, and all associated properties that will be used in the media session. The diagram shows that Peer1 will send the SDP offer and ICE candidates to the server and the server will pass that to Peer2. Peer2 will send the SDP answer and ICE candidates to the server and the server will pass that to Peer1.

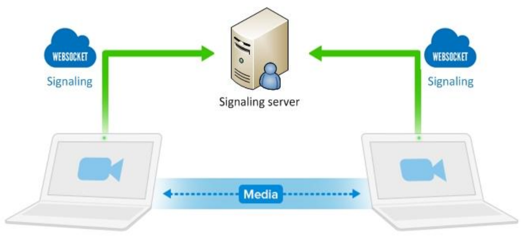


Figure 5. Signaling process

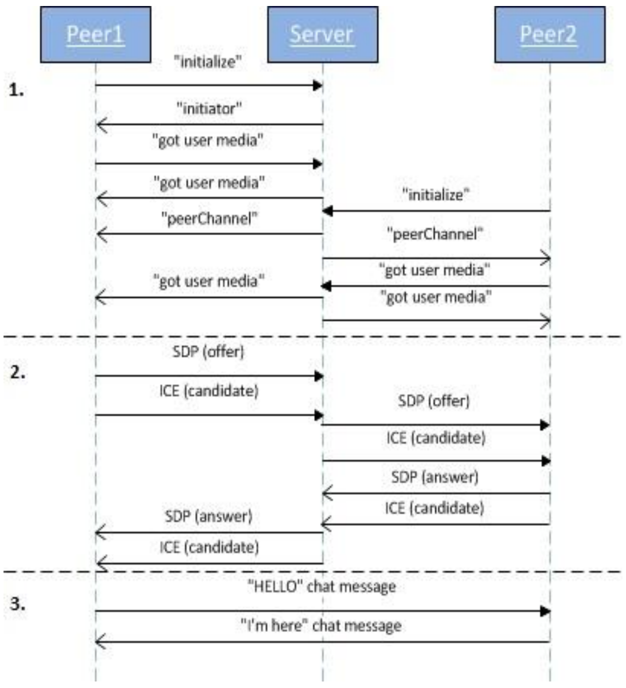


Figure 6. WebRTC message sequence chart

After this, the peer-to-peer connection is established. In addition, we have implemented a real-time text chat and now the peers can directly exchange messages without the server presence.

All signaling SDP and control messages, in our WebRTC application are transmitted as JSON (JavaScript Object Notation) objects (Figure 7.). JSON is a syntactic subset of JavaScript and has a great advantage of being natively interpreted by web browsers. JSON enables information exchange, as well as JSON objects generation and parsing.

In summary, there are three phases in Figure 6. The first phase – the peers are exchanging the control messages with the server, about the initiator, peer channel and user media. The second phase – creation of peer connection based on the SDP exchange. The third phase – the peer-to-peer connection is created and the peers can now exchange messages directly.

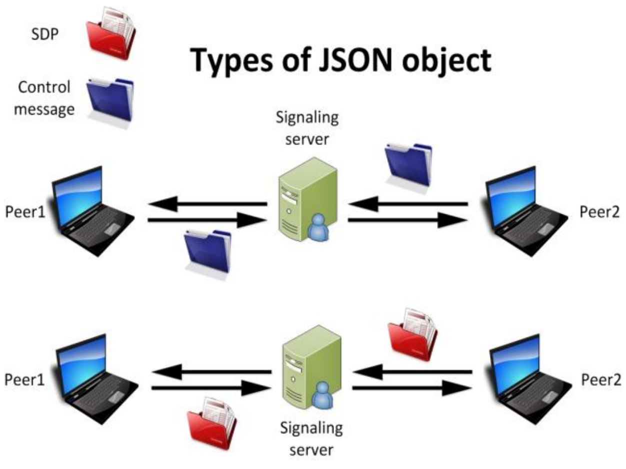


Figure 7. Types of JSON object

B. Server application

The server application is implemented as a WebSocket server written in Node and it can be used by any client or a server application. Node.js is an open source, cross-platform runtime environment for the server-side and networking applications. Node.js applications are written in JavaScript. It provides an event-driven architecture and a non-blocking I/O API that optimizes an application's throughput and scalability.

Our WebSocket server is using the WebSocket library. It is running on a localhost at port 2222 and it is available to anyone within the enterprise. The server implements the accepting, processing and forwarding message mechanism.

The WebSocket server is dealing with four types of control messages ("initialize", "initiator", "got user media" and "peerChannel"), two types of media information messages (SDP offer and SDP answer) and one type of network information messages (ICE candidate). Also it manages the number of clients, deciding whether a client is the initiator or should it be removed when the session is over.

C. Client application

The client application is implemented using the WebRTC API. The main task is to establish a peer-to-peer connection. Before a peer-to-peer connection gets established, the exchange of local and remote descriptions is performed (the audio and video media information).

In the following we will describe how the signaling offer and answer are exchange using the SDP. First, Peer1 runs the RTCPeerConnection createOffer() method. After that, it sets the local description using the setLocalDescription() method and then sends this session description to Peer2 via the signaling server. Then, Peer2 sets the description that Peer1 sent as a remote description using the setRemoteDescription() method. Peer2 runs the RTCPeerConnection createAnswer() method. In the createAnswer(), it sets the local description and sends it to Peer1. When Peer1 gets the Peer2 session description, it sets that as the remote description with the setRemoteDescription() method.

Figure 9. represents one snap shot of our application screen. Also there is the chat application extension, so peers can send text messages. For example, once installed within an enterprise network, our server (with URL http://localhost:2222/) will enable WebRTC peer-to-peer communication sessions.

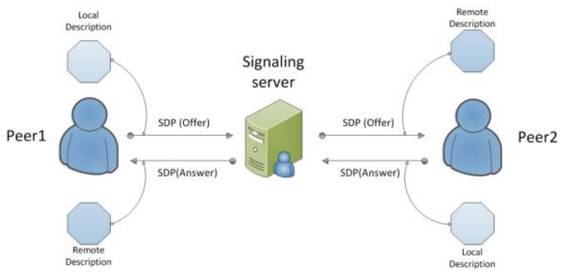
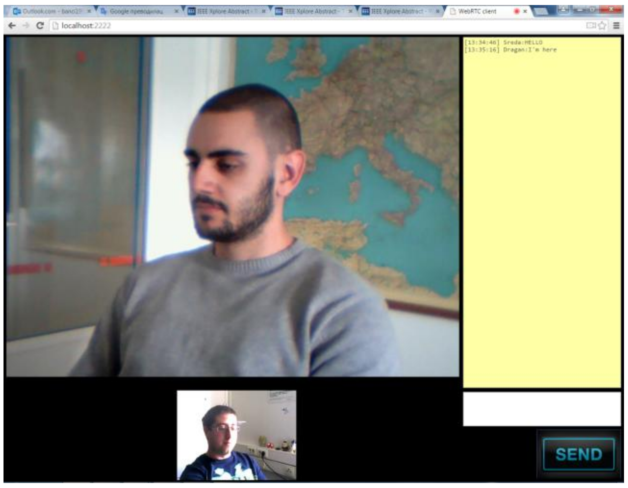


Figure 8. Offer/answer exchange between peers



```
[13:34:46] Sreda:HELLO
[13:35:16] Dragan:I'm here
```

Figure 9. Our application

III. CONLUSION

WebRTC is a powerful technology that benefits all areas of communication accessibility and feature rich tool. This technology has a potential to improve enterprise communications. It can provide a lot of benefits for enterprises. WebRTC technology reduces the costs of using VPN (Virtual Private Network) to connect remote branches and real-time communication is supported without the need for additional third-party software or plugins. Secure communication with employees in within an enterprise and its remote branches is enabled by state of the art encryption standards.

In the future we will develop an additional file sharing option and we will use encryption for both the media and signaling.

ACKNOWLEDGMENT

This work was partially supported by the Ministry of Education and Science of the Republic of Serbia under the project TR 36029.

REFERENCES

[1] A. Sergiienko, "WebRTC Blueprints, Develop your very own media applications and services using WebRTC," in Birmingham, Packt Publishing, ISBN 978-1-78398-310-0, May 2014.

[2] A. Zeidan, A. Lehmann, and U. Trick, "WebRTC enabled multimedia conferencing and collaboration solution," WTC 2014, in Berlin, Germany, Print ISBN 978-3-8007-3602-7, June 2014.

[3] W3C Editor's Draft, http://w3c.github.io/mediacapture-main/getusermedia

[4] K. Singh, A. Johnston, and J. Yoakum, "Taking on webRTC in an enterprise," Communications Magazine, IEEE, ISSN 0163-6804, April 2013.

[5] WebRTCstats, http://webrtcstats.com/webrtc-revolution-in-progress/

[6] R. Manson, "Getting Started with WebRTC, Explore WebRTC for real-time peer-to-peer communication," in Birmingham, Packt Publishing, ISBN 978-1-78216-630-6, September 2013.

[7] IETF - WebSocket, https://tools.ietf.org/html/rfc6455

[8] IETF - SDP, https://tools.ietf.org/html/rfc4566