On multi-user web conference using WebRTC

Cristian Cola
Technical University of Cluj Napoca
26-28, Baritiu St., 3400, Cluj-Napoca,
Email: colacristian@gmail.com

Honoriu Valean
Technical University of Cluj Napoca
26-28, Baritiu St., 3400, Cluj-Napoca
Email:Honoriu.Valean@aut.utcluj.ro

***Abstract*— The paper discuses the potential of the WebRTC in a multi-user video conference. All the signaling for multi-user video conferences is sustained by the XMPP server. Currently this technology is being deployed on Google Chrome, Opera and Firefox web-browsers. Without a standardized solution, service providers can implement various types of architectures. In this paper we propose a multi-video web conference solution that works in other browsers along with Firefox, Opera and Google Chrome. XMPP server is used as a signaling and transporting protocol.**

I. INTRODUCTION

Web Real Time Communication, WebRTC for short, enables one of the last major challenges of the web: the ability of humans to communicate via voice and video technologies. Integrating Real Time Communication (RTC) was difficult and time consuming; it required expensive audio and video technologies to be licensed or to be developed in house.

Google bought GIPS, a company that had developed many components required for RTC, opened sources and technologies that engaged with relevant standard bodies at the IETF and W3C. WebRTC has now implemented the open standards for real-time, plugin-free video, audio and data communication[6].

In this paper we evaluate a connection architecture using an XMPP server and the possibility of interoperating built in WebRTC, that is deployed for Google Chrome, FireFox and Opera browsers, and a custom plugin that we are building for other browser vendors that don't yet have the option to support real time communication.

We structured the remainder of the paper as follows: in section II we discussed the architecture of the system which is based on three parts 1) the XMPP server that handles the signaling and the data channel 2) the plugin that provides the support for other major browsers to operate in our system and 3) the peers that are made up of the users's browsers. We also describe the method that has been implemented and the connection flow of our existing system. In section III we carried out several tests that showed the differences between plugin initializations on multiple browsers and platforms. Than, we tested the video quality between the plugin and built in WebRTC and we tried to establish the CPU consumption on a "three user conference" between built in real time communication and a custom plugin. Section IV shows some related work, while section V, concludes the paper.

II. ARCHITECTURE

WebRTC developers can choose whatever messaging protocol they prefer, such as XMPP or any appropriate duplex (two-way) communication channel. The signaling process has not yet been standardized in WebRTC and is considered as a part of the application.

There are several options to solve the signaling: XHR/Comet, WebSockets, XMPP, etc.

WebSockets enable bidirectional sessions between the client (browser) and the web server. In this case, the messages that can be textual or binary can go from both directions. The disadvantage of this option is that not all the browsers support WebSockets.[3]

XHR/Coment is an option that enables a web server to send the messages to the clients - this is required for the signaling messages. This approach works on most web browsers and is easy to set up and use. The only problem is, that it does not support the presence feature.

The architecture proposed in this paper consists in three main components: *XMPP server*, that handles all the connections from the users and servers as a signaling part for other parties, the *peers*, that are made up of the users's browsers and the *plugin*, that represents the support for the displaying of the video conference in all major browsers.

A. XMPP server

XMPP defines a format for moving data between two or more communicating entities. Data exchanged over the XMPP is XML, XML giving the communication a rich, extensible structure. It is extremely easy to add new features to the protocol, which is both backwards and forwards compatible.[1]

We can use the XMPP protocol as transport protocol because it supports multi-user chat (MUC). XMPP will handle the session between multiple users that will join the video chat session. We can use the advantages of the built in presence handling. In our solution, the presence is used for signal joining the room and the possibility to start the P2P connection.

B. Plugin

In this section we discuss the ability to connect other browsers to our web conference model. We use JavaScript layered approach.

978-1-4799-4601-3/14/$31.00 ©2014 IEEE

1) Plugin JavaScript Connection: Since W3C and IETF and other browser vendors have not agreed to a final API protocol to implement WebRTC and to standardize a communication architecture, our solution for implementing multiuser conference for most browsers is to use a plugin. Our proposal is FireBreath. We compile libjingle library along side with Firebreath. Inside the compiled plugin we define similar methods to get "the user media" and creating the P2P connection similar with WebRTC built in browsers. The goal is to interoperate the browser plugin with the built in WebRTC that is deployed in Firefox, Opera and Google Chrome. To achieve this we need to create a JavaScript layered API that uses for Firefox, Opera and Google Chrome the methods that are defined on browser side. For the unsupported browsers, the plugin methods should used.

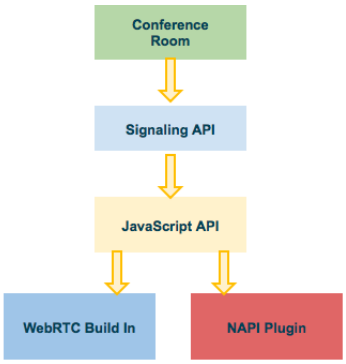

Fig. 1. Plugin Layered API used to communicate between WebRTC plugin and browser deployed plugin

As you can see in Fig. 1 we used a layered approach. The *JavaScript API* layer detects if the browser supports the built in WebRTC. If not, we should use the plugin that needs to be installed. The best solution to operate with two components (plugin and browser methods) is to use Lazy function definition pattern.

2) Plugin Architecture: The plugin has two components:

- plugin-engine
- plugin-renderer

plugin-engine - is responsible for getting the user media that is available on user's computer. After the user media is available the *plugin-engine* sets the media stream to the *plugin-renderer*. It will create a peer-to-peer connection and attach all the streams, local and remote, to the plugin-renderer and to P2P connection when other users's will be available.

plugin-renderer - is similar with a HTML5 video tag element. It is responsible for displaying the video streams from local devices or from other remote users that are connected to the conference.

C. Peers (users's browser)

Currently no major browser has a built in support for the XMPP protocol. By using some clever programming, you can tunnel XMPP sessions over HTTP connections efficiently and effectively. The technology that enables tunneling is called HTTP long polling [1].

Long polling is a traditional polling technique, but it allows emulating a push mechanism. With long polling, the client requests information from the server exactly as in normal polling but it issues HTTP requests at a much lower frequency. If the server does not have any information available for the client, instead of sending an empty response, the server holds the request open and waits for the response information to become available. When the data is available, the server sends the response to the client completing the open HTTP request.

Multiple libraries and protocol have been designed to take advantage of long polling. The XMPP's long polling is one of the oldest implementations ever developed. In XMPP this bridge is called BOSH (Bidirectional streams Over Synchronous HTPP) [1].

BOSH helps an HTTP client to establish a new XMPP session that transports stanzas back and forth over HTTP wrapped in a special *<body>* element. It also provides security features to make sure that XMPP sessions cannot be easily hijacked.

XMPP stanzas are the core part of the protocol. The applications that implement XMPP are concerned in sending and receiving multiple types of stanza elements.

A Stanza is an XML element which is well formed and completed. This element is sent from one entity to another over an XML stream that acts as an envelope for all the stanzas that are sent during a session. An XML stanza may contain child elements with some attributes in order to convey the desired information [1].

Each stanza is the first-level child element of the stream. It defines three types of stanza elements: *<presence/>*, *<message/>* and *<iq/>*.

The Strophe Library was created to make programming XMPP applications in JavaScript as easy as in any other language, hiding all the details of the managed connections.

Applications that are written with Strophe run on any of the current popular browsers. Strophe API is very simple, small and easy to use. It has numerous helpers, one of this is Strophe.Builder which helps you build stanzas quickly.

The *Signaling API* of layered approach (see Fig. 1) handles the communication between XMPP server and *JavaScript API*. All the signaling messages are handled by this layer and sent to *JavaScript API* that decides to assign the messages to plugin or built in WebRTC. When a user joins or leaves the chat room, *Signaling API* handles the P2P connection with other users involved in the conference.

D. XMPP signaling architecture

1) Connection flow: A video chat session is similar with a chat room. Each user that will join that room will have to get the list of the users that are present in that video room and create a peer connection with each individual user.

To create a peer connection we need to create a set of messages that is needed to be exchanged. Each message consists of an action that the user must do before it can join

the videoconference.

The actions are:

- session-initiate
- session-accept
- transport-info
- session-terminate

session-initiate - the user will initiate a peer connection. If the peer connection object is a success, then that peer connection will send an offer. This offer will be transported through the XMPP to a designated user with the action *session-initiate*. The offer contains a Session Description Protocol (SDP).

session-accept - the destination user will receive a message with the action *session-initiate* and it will create a peer connection, this peer connection will set the remote description with the SDP of the sender who sent the *session-initiate* action and then it will send an answer to the sender user.

transport-info - after the connection was established, *session-initiate* and *session-accept* messages were successfully sent, the peer connection will send some messages with a technique that is called Interactive Connectivity Establishment (ICE). This ICE message will be sent with XMPP protocol with the status *transport-info*.

session-terminate - this action will be sent to all the users that are connected to a video conference notifying them that a user will close all the connections. In this case all the peer connections with that user will be closed

Fig. 2 shows the flow to initiate a peer connection between two users that are connected to the XMPP server.

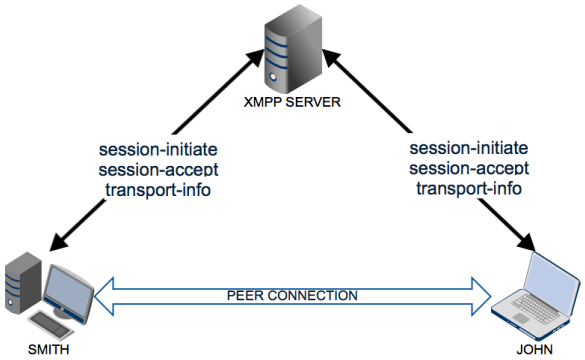

Fig. 2. XMPP signalling architecture

2) Functionality: After the peer connection was established, the transport for voice, video, data channel and information flow directly between the browsers. Fig.2 is called the triangle due to the shape of the signaling and media or data flow (base of triangle) [2].

Users Smith and John have to join the same video session (same room). User Smith notices that user John is available for videoconference and starts sending a message with the action *session-initiate*. After user John receives the *session-initiate* message it will send a message with the action *session-accept* informing user Smith that he can send the messages with the action flag *transport-info*. After this handshake has been successfully done, the peer connection can start between those two users. This process is done for each user that joins the video session. This technique is also known as full mesh or fully distributed conferencing architecture[4].

For each user that joins the session it will establish a full mesh of peer connections with other users and it will have the video streams from other users that will be streamed in a different HTML video tag element with some appropriate labelling. As new users join the session new peer connections are established to send and receive media streams.

III. Performance Evaluation

We carried out several tests and we outlined some differences between our plugin implementation and built in WebRTC. We tested the plugin in several known browsers: Internet Explorer 11, Safari, FireFox, Chrome and Opera. Our tests consist in plugin initialization on different browsers and systems and CPU consumption by the process browser that is needed in a ”three user conference”.

The tests were carried out on the following configurations:

- **Mac OS** - Dual Intel Xeon Quad Core CPUs at 2.8 Ghz with 16Gb of RAM, Mac OS X Ver. 10.9.3
- **Windows** - Intel Core i7 at 2.3 Ghz with 8 Gb of RAM, Windows 8.1

A. Initialization / CPU consumption

In this section we measured the delay time for initializing the *plugin-engine* component.

Table I shows the time required for the plugin to be initialized in the most frequently used browsers.

TABLE I
PLUGIN INITIALIZATION ON MAJOR BROWSERS

Browser	Mac OS	Windows
Chrome	586ms	270ms
Firefox	800ms	290ms
Internet Explorer 11	N/A	402ms
Safari	860ms	336ms
Opera	566ms	285ms

Next, we compared the CPU consumption for built in WebRTC and our plugin in a ”three user conference” in all major browsers. We measured the percentage of the browser process used to join the conference.

Table II shows the CPU consumption test carried out on a Windows machine.

B. Video Quality

WebRTC supports resolutions up to 720p. To avoid CPU bottlenecks we reduced the resolution for WebRTC build in

TABLE II
CPU CONSUMPTION IN A THREE USER CONFERENCE

Browser	Plugin	WebRTC
Chrome	1.3%	38%
FireFox	1.5%	30%
Internet Explorer 11	25.5%	N/A
Safari	4.1%	N/A
Opera	3.1%	35%

and plugin to 640x480. The audio/video quality may depend on the connection bandwidth and the number of the participants in the conference.[5]

Fig. 3. Video quality on Plugin vs. WebRTC

Fig. 3 shows the video quality of the conference. The picture on the left, is rendered with *plugin-renderer* component while the picture on the right is rendered with HTML5 video tag element. This video quality test was carried out on Google Chrome browser on a Mac platform.

IV. Related Work

Many researches and solution proposals have been conducted in the WebRTC area, most of the projects were proof-of-concept.

- Bistri describes the solution of simple WebRTC video chat implementation. It choses to use XMPP as signaling protocol. The first implementation was made in Flash and then they migrated the project to WebRTC when the W3C and IETF first drafted the WebRTC specifications.

- Tringgr is a WebRTC powered video chat that uses WebSockets as signaling protocol. The solution implemented supports up to 4 participants in a conference.

- Skype is a stand-alone application which is compatible with all major operating systems (Windows, Linux, MacOS) and mobile operating systems (Android, iOS, Windows). This enables voice, IM and video communication. It's main advantage is that it has a huge installation base (approx 299 million users). It supports native PTSN (Public Switched Telephone Network) and integrates with an enterprise calling platform called Lync. The WebRTC's main advantage is that the technology is an open source and it was designed to be tied to the browser. This means, that the users will not have to download anything to take advantage of this technology.

V. Conclusion / future work

In this paper we carried out a scenario of connecting multiple users into one web conference using built in WebRTC and a custom plugin for other browsers that currently did not adopt this technology.

The main advantage of plugin integration for WebRTC is the possibility to custom the video frame and the User Interface. For example, we can have the feature of selecting the input and output devices right inside the browser and we can test the sound quality with a simple implementation of a volume unit meter. Fig. 4 shows the drop-downs of all the devices that are available on the user's system and the ability to test the sound quality.

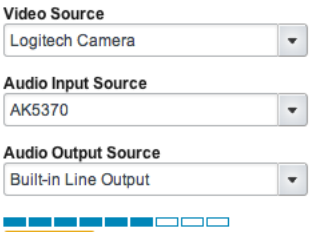

Fig. 4. Simple implementation of input/output selection devices and volume unit meter

The major disadvantage is that the user must download and install the plugin in order to have access to our system from other browsers than Firefox, Opera and Google Chrome. The plugin initialization and creating the peer-to-peer connection could take some efforts on a user's system. This may depend on the number of participants that join the conference.

One of the benefits of this technology is accessibility. If the user navigates on one of the supported browsers, it should be simple to take part in a video conference. The user would not have to download any applications, plugins, updates, etc.

We are constantly involved in research and implementation efforts to find better solutions to implement WebRTC in other systems than the supported ones. Our goal is to implement this WebRTC conference solution in mobile systems.

We are following the research activities of W3C and IETF as well as the browser vendors's implementation. The first version of the standards in the IETF and W3C is expected to be published in 2014. Web browsers implementations are expected to be available sometime in late 2014.

References

[1] J. Moffitt, *Professional XMPP Programming with Javascript and jQuery*, Wiley Publishing, Inc., Indianapolis, Indiana, 2010, pp. 4-5, 30-36.
[2] A. B. Johnston and Daniel C. Burnett, *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, Digital Codex LLC P.O. Box 9131, St. Louis, USA, 2013, pp. 5-9, 16-22.
[3] R. Manson, *Getting Started with WebRTC*, Packt Publishint Ltd. Livery Place, UK, 2013.
[4] I. Grigorik, *High Performance Browser Networking*, O'Reilly Media, 2013, chapter 18.
[5] V. Singh and A. A. Lozano and J. Ott, *Performance Analysis of Receive-Side Real-Time Congestion Control for WebRTC*, Finland 2013.
[6] HTML5, http://www.html5rocks.com/en/tutorials/webrtc/basics/ (as visited 05/03/2012)