WebRTC using JSON via XMLHttpRequest and SIP over WebSocket
Initial Signalling Overhead Findings

Michael Adeyeye[1], Ishmeal Makitla[2] and Thomas Fogwill[2]

[1]*Department of Information Technology, Cape Peninsula University of Technology, Cape Town, South Africa*
[2]*The Next Generation Network and Architecture Research Group, CSIR, Pretoria, South Africa*

Keywords: WebRTC, IMS, SIP, Browser Communication.

Abstract: Web Real-Time Communication (WebRTC) introduces real-time multimedia communication as native capabilities of Web browsers. With the adoption of WebRTC the Web browsers will be able to use WebRTC to communicate with one another (peer-to-peer), and with WebSocket servers such as Mobicents SIP Servlets and other server technologies that support WebSocket communication to enable SIP-to-WebRTC communication. This position paper discusses the two common methods of doing real-time communication in Web browsers through WebRTC. The methods are JavaScript Object Notation (JSON) via XMLHttpRequest (XHR) and Session Initiation Protocol (SIP) via WebSocket. A three-user WebRTC video chat prototype application was developed and used to evaluate both methods. Additional signalling overhead introduced into a browser by each method was determined. The results showed WebRTC-SIP/WS has more overhead than WebRTC-JSON/XHR. This signalling overhead findings are useful in informing the WebRTC working groups in terms of additional overhead introduced by proposed WebRTC methods, the finding could also help application developers make decision on their choice of technologies and protocols when developing WebRTC-supported applications.

1 INTRODUCTION

The Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) are currently tasked with bringing WebRTC among browsers to an acceptable level in both the industry and the academia. WebRTC is an open framework that offers web application developers the ability to write rich real-time multimedia applications (e.g. video and gaming applications) on the web without requiring plugins or extensions. Its purpose is to help build a strong Real Time Communication (RTC) platform that works across multiple web browsers and platforms. In an implementation, the WebRTC API will abstract several key components for real-time audio, video, networking and signal (WebRTC. (2011a); WebRTC. (2011b)). While the IETF is standardizing the signalling protocols and media technologies (e.g. codecs) required in WebRTC, the W3C is standardizing the APIs and browsers for real time communication.

There are many implementations of RTC among web browsers (Linner et al., 2010; SIP on the Web, 2012; SIP-JS, 2012; The Phono WebRTC, 2012; Adeyeye et al., 2012) and the WebRTC itself (Ericsson WebRTC, 2012; Chrome WebRTC Implementation, 2012; RTCWeb-SIP, 2011). The standard signalling protocol for WebRTC is JavaScript Session Establishment Protocol (JSEP), however, Remote Object Access and Replication (ROAR) has also been used in some existing implementations. The reason JSEP is the preferred protocol is because it moves control or negotiations from a browser to JavaScript (in an application). In addition, there is a need to make the WebRTC implementation look similar to the SIP Offer and Answer. Although the VP8 codec seemed to be the preferred codec for WebRTC, it is faced with royalty problems. Hence, codecs for the WebRTC are also being addressed.

WebRTC is built on the PeerConnection API and represents what browser vendors will implement and expose to web application developers. Web application developers can choose an underlying protocol depending on their project requirements. The underlying protocols, also called the sub-protocols, include SIP and XMPP (with Jingle). The Libjingle library, like a SIP stack, supports (Session Transversal Utilities for NAT (STUN) and Transversal Using Relays

and NAT (TURN). Both these Interactive Connectivity Establishment (ICE) techniques, namely STUN and TURN, make communication possible when the communicating endpoints are behind a firewall.

The motivation for this work is that application developers will soon begin to create innovative WebRTC-supported applications with little or no consideration on the total cost of usage of their applications. An application with a high signalling overhead would incur more cost with poorer quality of experience for users having low Internet bandwidth and paying high cost for Internet access. This work examined the additional signalling overhead introduced by WebRTC applications. The contribution of this research is therefore the development of a three-user WebRTC video chat application with a report on the signalling overheads introduced by the two common methods of doing WebRTC.

The remainder of this paper is arranged as follows: Section II discusses the common methods of doing WebRTC within compliant Web browsers and the current ways of implementing video streaming using WebSocket. Section III presents the three-user WebRTC vide chat prototype which was used to evaluate the resultant signalling overhead. Section IV then presents and discusses the resultant signalling overhead of the two commond methods of doing WebRTC. In Section V the paper is concluded.

2 WEB REAL-TIME COMMUNICATION (WebRTC) METHODS AND ISSUES

The two prominent ways of doing WebRTC are using pure SIP via websocket (WebRTC-SIP/WS) and JavaScript Object Notation via XMLHttpRequest (WebRTC-JSON/XHR). While the former uses a WebRTC-SIP proxy/gateway as its application engine and SIP over websocket for signalling, the latter uses a custom engine (e.g. the Google App. Engine) as its application engine and JSON over XHR for signalling. For the Google App. Engine, the JSON/XHR signalling is done via its Channel API. However, both approaches are based on JSEP, which mimics the SIP Offer and Answer signalling.

There are however other implementations developed to meet specific requirements. An example is the Ericsson WebRTC implementation (Ericsson WebRTC, 2012), which uses ROAR. In this example, some changes were made to the webkit libraries in the Epiphany web browser in order to support WebRTC. There are other kinds of implementation in the

form of an extension to a browser. An example is the IEWebRTC extension (which uses ChromeFrame) for Internet Explorer (webrtcie_ext, 2012). As web browsers are being extended, the number of WebRTC applications and frameworks, such as SIPML5 (which uses SIP over websocket) (sipml5, 2012) and SIP-JS (with support for Flash-network) (SIP-JS, 2012), are rapidly increasing. At the time of this research, Google Chrome is taking the lead in the WebRTC implementation. Mozilla Firefox is yet to have a version that has the PeerConnection or getUserMedia API. A SIP stack (called SIPCC) is now being integrated into it (Ikran - FF_as_SIP_endpoint, 2012). Hence, it does not currently support WebRTC. Other browser makers, such as Microsoft and Opera, are also contributing to the WebRTC standadization.

Figure 1 shows the signalling between two UAs (User Agents) or devices; the sequence of events starts from top to bottom. Some of the processes (such as PeerConnectionFactory, ProcessSignalingMessage and OnSignalingMessage) are peculiar to Google Chrome, which uses the libjingle. A caller first creates a new peerconnection and adds stream using the PeerConnection API as shown in Figure 1. In addition, a local session description (for audio and/or video) is applied. ICE is then started in order to get available IP (Internet Protocol) address and port number for media transfer (these additional processes are not shown for simplicity). A peerconnection and remote session description (for the callee) are later created. When a callback at the callee's notifies that a stream is added (via a channel), an offer is created. It is sent and processed by the caller. An answer is then sent back and a local session description (for the callee) is created. The answer, which contains the remote session description and some hints, is sent to the callee. A local session description (for audio or/and video) is then set and applied in the callee's browser. Lastly, ICE is also started in order to get available IP (Internet Protocol) address and port number for media transfer. The caller later applies the remote session description in order to present video/audio from the callee. Encryption of WebRTC media in Google AppEng is achieved by sending the UDP (User Datagram Protocol) data via SCTP (Stream Control Transmission Protocol) and DTLS (Datagram Transport Layer Security).

On the other hand, there are still NAT and firewall issues in the WebRTC. In addition, SBCs (Session Border Controllers) could be required to handle connections between two or more domains. ICE techniques (STUN and TURN) are likely to increase duration to set-up a call. Security of media and permissions are also hot issues in the WebRTC, though
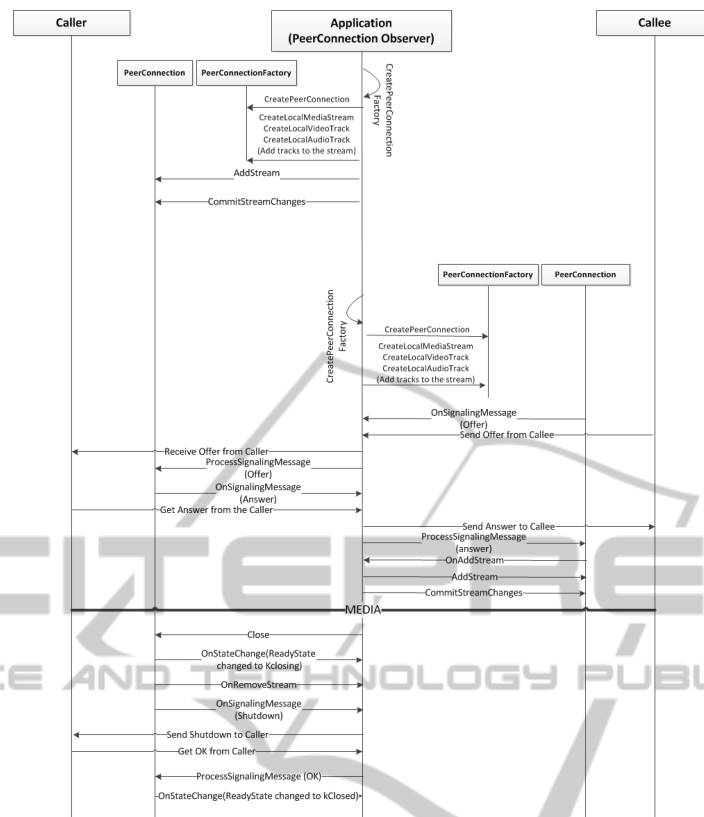
Figure 1: The WebRTC Signalling in a Call Session.

there are a couple of solutions that can be used. Other issues of interest include recording video, supporting other SIP/SIMPLE features, such as presence and messaging, and doing multi-user video chat using the WebRTC framework.

As of this writing, and whereas the issues mentioned above are receiving sufficient attention, the issue of additional overhead introduced by WebRTC methods has not been investigated. The next section describes a prototype application which was used to investigate additional signalling overhead.

3 A THREE-USER WebRTC VIDEO CONFERENCE PROTOTYPE

To determine the additional signalling overhead introduced by WebRTC applications, a three-user WebRTC video chat application was developed as a prototype for this research using the PeerConnection API. The prototype application used both WebRTC communication methods discussed in Section III namely WebRTC-SIP/WS and WebRTC-JSON/XHR.

Google Chrome Web browser which integrated libjingle (with XMPP) was used for experimentation as it is the only browser with an acceptable level of WebRTC support required for this research. As at the time of writing, this is the only work that has considered signalling overhead introduced by the two WebRTC methods, In addition, the three-user WebRTC video chat application is one of the few WebRTC video chat applications that support three or more users. Most WebRTC video chat applications are only for two users, since WebRTC is currently being standardized. The Three-user WebRTC video chat is depicted in Figure 2.

Although the application was built for no more than three users (as shown in Figure 2), the signalling between two users is shown here (Figure 1) for simplicity reasons. The application in-between the two User Agents (UAs) acts as a B2BUA and is common feature among multi-user conference applications. The video conference application was first developed and deployed in Google AppEngine (i.e. the WebRTC-JSON/XHR). It used the Channel API in the Google AppEngine for WebRTC signalling and the getUserMedia and PeerConnection APIs in the Google Chrome browser for media streaming. Since
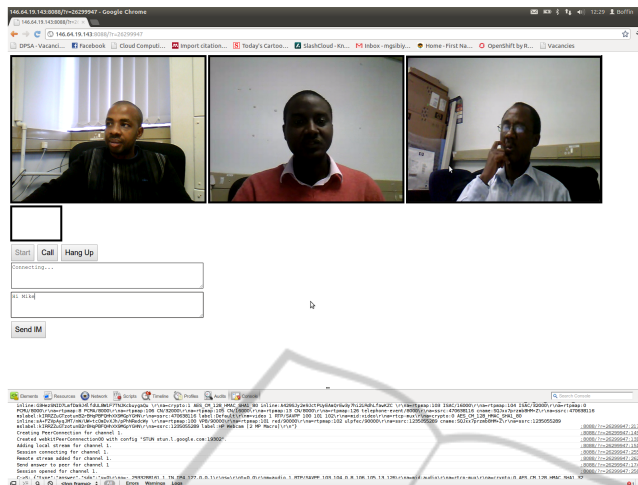
Figure 2: The Three-user WebRTC demo.

the PeerConnection API only works for two devices, each device created two instances of "webkitPeer-Connnection00" and each instance was used to set-up a peer-to-peer connection with the other device. In order to demonstrate WebRTC-SIP/WS, a SIP servlet application was modified and deployed into the Mobicents AS (Application Server) as a SIP proxy in an IMS. The Mobicents SIP Servlets AS used Apache Tomcat 7.0, which supports WebSocket. The SIP proxy acted as a B2BUA, which sets up a video chat among the three users. The source of the application is published on the Internet for contributions from interested parties and the Open Source (OS) community [project-source]. It is one of the few WebRTC works on the Internet that support more than two users. Figure 2 also shows the signalling in a browser using the browser's developer tools.

4 WebRTC-JSON/XHR AND WebRTC-SIP/WS SIGNALLING OVERHEAD

As stated in Section II, the issue of signalling overhead introduced by the two WebRTC methods has not been studied before. Therefore, in order to report the performances and differences between WebRTC-JSON/XHR and WebRTC-SIP/WS method of doing WebRTC, an experiment was performed using each method. The signalling overhead in a peer-to-peer connection was measured. The upload and download speed for the network were 0.15Mbps and 0.81Mbps, respectively. The test was carried out on a Local Area Network (LAN), and the WebRTC-SIP/WS application played the role of both a WebRTC-SIP proxy/gateway and a SIP Registrar. Hence, there were no outbound connections. Like every application, its QoS (Quality of Service) depends on the network speed. Connection time (latency) and signalling overheads are two factors that can be used to evaluate the performance of the two WebRTC methods. The connection time and delay were determined by running Network Time Protocol (NTP) on all machines used in the experiment. While connection time among peers in a video chat was infinitesimal or not noticeable (being a test performed on a LAN), the signalling overhead was noticeable. As a result, this work focuses on the signalling overheads of each WebRTC method. The payload of each application was not included in the values of signalling overheads. Table 1 shows the signalling overheads in a web browser when the browser runs the WebRTC prototype applications for the three-urse video chat. In addition, the values were compared with overheads introduced by a regular SIP client - PJSIP. The result shows the signalling overheads as they increases in both WebRTC approaches. The experiment was repeated multiple times in order to report mean values and, for each value, its variance in brackets for the overheads.

As reported on Table 1, all results show a limited variance. A basic HTTP request-response (with no payload) is 150B. The HTTP overhead is higher than the WebRTC-JSON/XHR overhead for a completed session (104B) because the HTTP server (Apache) responded with some additional information in its response header. It is however possible for a developer to compress HTTP response headers or reduce the response information to the essential ones. The WebRTC-SIP/WS overhead can affect quality of experience, where access to the Internet is costly and the Internet connection speed is low.

Table 1: Signalling Overhead in WebRTC via JSON/XHR and SIP/WS.

	WebRTC-JSON/XHR	WebRTC-SIP/WS	A SIP Client (PJSIP)
On Register	13B (0.58)	34B (1)	2.5kB (1.1)
On Invite	39B (0.58)	204kB (0.88)	4.9kB (1.02)
During Call Session	78B (0.78)	204kB (1)	9.6kB (1)
Ending Call Session	104B (0.78)	275kB (1)	10.4kB (1.02)

Shown in Table 1 are initial signalling overhead findings from the experiment that was conducted. These results open up a number of issues such as why is there such a big difference (i.e. 39B versus 204KB) between WebRTC-JSON/XHR and WebRTC-SIP/WS. Could this be the fact that SIP is XML based and that XML uses too many bytes simply to structure its content (i.e opening and closing tags) which may account for this big difference in byte-sizes? These and many other issues we hope to investigate as part of our continued WebRTC experimental research.

a XULRunner or Chrome Application) thereby directly communicating with the underlying protocols and mechanisms in that browser.

Interoperability between WebRTC and current SIP servlets and VoIP services have great potential to create new markets. Necessarily current efforts within IETF and other working groups for WebRTC seek to address WebRTC-SIP interoperability. This means that further experiments and analyses of potential signalling overhead that this will introduce are very curcial to inform the direction taken by these WebRTC working groups.

5 CONCLUSIONS

A three-user WebRTC video chat has been developed and released to the OS community and researchers exploring WebRTC. In addition, the signalling overheads for the two WebRTC approaches have been reported.The support for WebRTC would create an additional ways of communicating between two devices. Voice services in existing telecommunication networks may likely drop as customers will pay more for data services in order to use WebRTC. Like there are unique attributes in CSS for different browsers, developers may need use some browser-specific features, most notably in JavaScript, after the standardization effort. They would have to choose what approach they want to use to develop their applications, and one of their considerations would be the signalling overhead.

With a Web browser becoming a real-time communication application, its installer file size is expected to drastically increase as it will now support new features, such as WebRTC and WebGL. While libjingle (with XMPP) is integrated into Google Chrome, a SIP stack is integrated in Mozilla Firefox to implement WebRTC. The integration of these protocols would open enormous opportunities for developers. On one hand, web developers can develop websites and applications that would run in a browser using HTML5 with the APIs exposed to webpages. On the other hand, application developers can develop applications that work with a browser internals (e.g.

REFERENCES

WebRTC, http://www.webrtc.org, accessed on October 13, 2011.

IETF WebRTC, http://tools.ietf.org/wg/rtcweb, accessed on October 13, 2011.

David Linner, Horst Stein, Ulrich Staiger and Stephan Steglich, "Real-time Communication Enabler for Web 2.0 Applications," in: Proceedings of the Sixth International Conference on Networking and Services (ICNS '10), Cancun, Mexico, March 7-13, 2010, pp. 42 - 48.

SIP on the Web, http://sip-on-the-web.aliax.net, accessed on June 11, 2012.

SIP-JS, http://code.google.com/p/sip-js, accessed on June 11, 2012.

The Phono WebRTC, http://phono.com/webrtc, accessed on June 11, 2012.

Michael Adeyeye, Neco Ventura and Luca Foschini, "Converged Multimedia Services in Emerging Web 2.0 Session Mobility Scenarios ," in: the Springer Wireless Networks (WINET) Journal. DOI: 10.1007/s11276-011-0394-z.

Ericsson WebRTC, https://groups.google.com/group/ericsson-labs-web-rtc, accessed on June 11, 2012.

Chrome WebRTC Implementation, http://www.w3.org/2011/04/webrtc/wiki/images/7/7f/Webrtc-chrome-impl-status.pdf, accessed on June 11, 2012.

IETF RTCWeb-SIP WG, http://tools.ietf.org/html/draft-kaplan-rtcweb-sip-interworking-requirements-01, accessed on October 13, 2011.

The IMS World Forum Summary, http://www.alanquayle.com/blog/2012/04/the-ims-world-

forum-summary-pa.html, accessed on June 11, 2012.

WebRTC for IE, http://code.google.com/p/webrtc4ie/, Accessed on January 17, 2012.

SIPML5, http://www.sipml5.org/, accessed on June 11, 2012.

FF as SIP Endpoint, https://github.com/ethanhugg/ikran, Accessed on January 17, 2012.

Three User WebRTC Chat Source, https://github.com/ micadeyeye/three-user-webrtc, Accessed on August 3, 2012.

Vijay K. Gurbani, Xian-He Sun and A. Brusilovsky, "Inhibitors for Ubiquitous Deployment of Services in the Next-Generation Network," in: the IEEE Communications Magazine, Vol. 43, No. 9, pp. 116-121, September 2005.

Karim Sbata, Houda Khrouf, Sabine Zander and Monique Becker, "Converging Web and IMS Services: Stakes and Solution Proposals," in: Proceedings of the International ACM Conference on Management of Emergent Digital EcoSystems (MEDES '09), Lyon, France, October 27-30, 2009.

Haruno Kataoka, Masashi Toyama, Yoshiko Sueda, Osamu Mizuno and Kenji Takahashi, "Demonstration of Web Contents Collaborative System for Call Parties," in: Proceedings of the 7th IEEE Consumer Communications and Networking Conference (IEEE CCNC '10), Las Vegas, Nevada, USA, January 9-12, 2010.

Google Chrome Frame, http://www.google.com/ chromeframe?quickenable=true, Accessed on January 17, 2012.

Google WebRTC Samples, http:// code.google.com/p/ webrtc-samples/, Accessed on January 17, 2012.

Google AppRTC, https://apprtc.appspot.com, Accessed on January 17, 2012.

Hideo Nishimura, Hiroyuki Ohnishi and Miki Hirano, "Architecture for Web-IMS Co-operative Services for Web Terminals ," in: Proceedings of the 13th International Conference on Intelligence in Next Generation Networks (ICIN '09), Bordeaux, France, October 26 - 29, 2009, pp 1-6.

The PJSIP Project, http://www.pjsip.org, April 12, 2012.

The Mozilla Firefox Web browser, http://www.mozilla.org, April 12, 2012.

M. Handley and V. Jacobson, "SDP: Session Description Protocol," IETF RFC 2327, April 12, 2012.