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

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

[1] Cape Peninsula University of Technology, Cape Town, South Africa
adeyeyem@cput.ac.za
[2] CSIR Meraka Institute, Pretoria, South Africa
{imakitla, tfogwill}@csir.co.za

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. 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 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 [1] [2]. While the IETF is standardizing the signaling 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 [3] [4] [5] [6] [7] and the WebRTC itself [8] [9] [10]. 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.

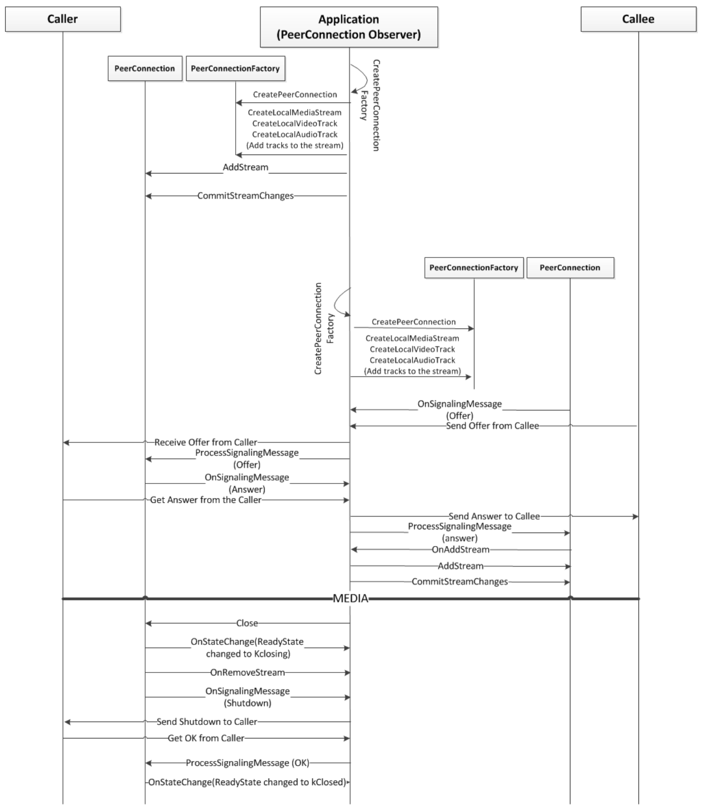
The motivation for this work is that application developers often create innovative WebRTC-supported applications with little consideration for the usage cost of their applications. An application with a high signalling overhead would incur more cost with a poorer quality of experience for users having poor and expensive Internet access. This work examined the signalling overhead introduced by WebRTC applications. Its contribution is the development of a three-user WebRTC video chat application with a report on the signalling overheads of the two common methods of doing WebRTC.

The remainder of this paper is arranged as follows: Section 2 discusses the common methods of doing WebRTC within compliant Web browsers and the current ways of implementing video streaming using WebSocket. Section 3 presents the three-user WebRTC vide chat prototype which was used to evaluate the resultant signalling overhead. Section 4 then presents and discusses the resultant signalling overhead of the two commond methods of doing WebRTC. In Section 5 the paper is concluded.

2 Web Real-Time Communication (WebRTC) Methods and Issues

The two prominient 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 [8], 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 Chrome-Frame) for Internet Explorer [11]. As web browsers are being extended, the number of WebRTC applications and frameworks, such as SIPML5 (which uses SIP over websocket) [12] and SIP-JS (with support for Flash-network) [5], 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 [13]. Hence, it does not currently support WebRTC. Other browser makers, such as Microsoft and Opera, are also contributing to the WebRTC standadization.



Caller
Application (PeerConnection Observer)
Callee
PeerConnection
PeerConnectionFactory
CreatePeerConnection Factory
CreatePeerConnection
CreateLocalMediaStream
CreateLocalVideoTrack
CreateLocalAudioTrack
(Add tracks to the stream)
AddStream
CommitStreamChanges
PeerConnectionFactory
PeerConnection
CreatePeerConnection Factory
CreatePeerConnection
CreateLocalMediaStream
CreateLocalVideoTrack
CreateLocalAudioTrack
(Add tracks to the stream)
OnSignalingMessage (Offer)
Send Offer from Callee
Receive Offer from Caller
ProcessSignalingMessage (Offer)
OnSignalingMessage (Answer)
Get Answer from the Caller
Send Answer to Callee
ProcessSignalingMessage (answer)
OnAddStream
AddStream
CommitStreamChanges
MEDIA
Close
OnStateChange(ReadyState changed to Kclosing)
OnRemoveStream
OnSignalingMessage (Shutdown)
Send Shutdown to Caller
Get OK from Caller
ProcessSignalingMessage (OK)
OnStateChange(ReadyState changed to kClosed)

Fig. 1. WebRTC Signalling in a Call Session

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 callees 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 callees 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 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 the time required to set-up a call. Security of media and permissions are also hot issues in the WebRTC, though 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.

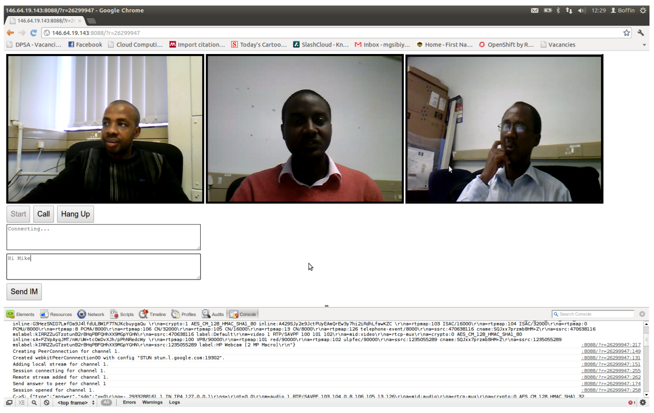
Whereas the issues mentioned above are receiving sufficient attention, the issue of overhead introduced by WebRTC has not been investigated. The next section describes a prototype application which was used to investigate signalling overhead.

3 A Three-user WebRTC Video Conference Prototype

To determine the 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.

The application was built for up to three users (as shown in Figure 2). For simplicity, the signalling between only two users is shown in Figure 1. 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 the PeerConnection API only works for two devices, each device created two instances of webkitPeerConnnection00 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 [14]. 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 browsers developer tools.



```
inline:G3HezSNID7LmfDa9J4lfdUL8W1F7TNJKcbuygaQu \r\na=crypto:1 AES_CM_128_HMAC_SHA1_80 inline:A429SJy2e9JctPUyEAmQrEw3y7hi2iRdhLfawKZC \r\na=rtpmap:103 ISAC/16000\r\na=rtpmap:104 ISAC/32000\r\na=rtpmap:0
PCMU/8000\r\na=rtpmap:8 PCMA/8000\r\na=rtpmap:106 CN/32000\r\na=rtpmap:105 CN/16000\r\na=rtpmap:13 CN/8000\r\na=rtpmap:126 telephone-event/8000\r\na=ssrc:470638116 cname:SQJxx7przmb8HM+Z\r\na=ssrc:470638116
mslabel:kIRRZZuGTzotunB2rBHqPBFQHhXX9MGpYGHN\r\na=ssrc:470638116 label:Default\r\nm=video 1 RTP/SAVPF 100 101 102\r\na=mid:video\r\na=rtcp-mux\r\na=crypto:0 AES_CM_128_HMAC_SHA1_80
inline:sA+FZVpAyqJMT/nW/UW+tcOmDvXJh/pPhNRedcWy \r\na=rtpmap:100 VP8/90000\r\na=rtpmap:101 red/90000\r\na=rtpmap:102 ulpfec/90000\r\na=ssrc:1235055289 cname:SQJxx7przmb8HM+Z\r\na=ssrc:1235055289
mslabel:kIRRZZuGTzotunB2rBHqPBFQHhXX9MGpYGHN\r\na=ssrc:1235055289 label:HP Webcam [2 MP Macro]\r\n"}                                    :8088/?r=26299947:217
Creating PeerConnection for channel 1.                                                                                                 :8088/?r=26299947:149
Created webkitPeerConnnection00 with config "STUN stun.l.google.com:19302".                                                             :8088/?r=26299947:131
Adding local stream for channel 1.                                                                                                     :8088/?r=26299947:151
Session connecting for channel 1.                                                                                                      :8088/?r=26299947:255
Remote stream added for channel 1.                                                                                                     :8088/?r=26299947:262
Send answer to peer for channel 1                                                                                                      :8088/?r=26299947:174
Session opened for channel 1.                                                                                                          :8088/?r=26299947:258
C->S: {"type":"answer","sdp":"v=0\r\no=- 2933288161 1 IN IP4 127.0.0.1\r\ns=\r\nt=0 0\r\nm=audio 1 RTP/SAVPF 103 104 0 8 106 105 13 126\r\na=mid:audio\r\na=rtcp-mux\r\na=crypto:0 AES_CM_128_HMAC_SHA1_32
```

Fig. 2. Three-user WebRTC demonstration

4 WebRTC Signalling Overhead

As stated in Section 2, 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 I 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.

	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)	240KB (1)	9.6kB (1)
Ending Call Session	104B (0.78)	275KB (1)	10.4kB (1.02)

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

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 slow.

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.

5 Conclusion

A three-user WebRTC video chat has been developed and released to the OS community and researchers. 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. 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. 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.

References

1. WebRTC project: WebRTC. http://www.webrtc.org (2011) Accessed on 13 Oct0ber 2011.
2. IETF WebRTC WG: Rtcweb Status Pages: Real-Time Communication in WEB-browsers (Active WG). http://tools.ietf.org/wg/rtcweb (2011) Accessed on 13 October 2011.
3. Linner, D., Stein, H., Staiger, U., Steglich, S.: Real-time communication enabler for web 2.0 applications. In: Sixth International Conference on Networking and Services (ICNS10), Cancun, Mexico (2010) 42 – 48
4. Millan, J.L., Castillo, I.B.: Sip on the web. http://sip-on-the-web.aliax.net (2012) Accessed on 11 June 2012.
5. SIP-JS project: SIP-JS. http://code.google.com/p/sip-js (2012) Accessed on 11 June 2012.
6. Phono.com: The phono webrtc. http://phono.com/webrtc (2012) Accessed on 11 June 2012.
7. Adeyeye, M., Ventura, N., Foschini, L.: Converged multimedia services in emerging web 2.0 session mobility scenarios. Wireless Networks (WINET) **18**(2) (2012) 185–197
8. Ericsson Labs: Ericsson Labs WebRTC. https://groups.google.com/group/ericsson-labs-web-rtc (2012) Accessed on 11 June 2012.
9. Alvestrand, H.: WebRTC API in Chrome: implementation experience. http://www.w3.org/2011/04/webrtc/wiki/images/7/7f/Webrtc-chrome-impl-status.pdf (2012) Accessed on 11 June 2012.
10. IETF RTCWeb-SIP WG: Requirements for Interworking WebRTC with Current SIP Deployments. http://tools.ietf.org/html/draft-kaplan-rtcweb-sip-interworking-requirements-01 (2011) Accessed on 11 June 2012.
11. WebRTC4ie project: WebRTC for Microsoft Internet Explorer. http://code.google.com/p/webrtc4ie/ (2012) Accessed on 11 June 2012.
12. SIPML5: World's first HTML5 SIP client. http://www.sipml5.org/ (2012) Accessed on 11 June 2012.
13. Ikran: An addon for Mozilla for making audio/video call using SIP. https://github.com/ethanhugg/ikran (2012) Accessed on 17 January 2012.
14. Adeyeye, M., Makitla, I.: Three-User-WebRTC Video Chat Demo. https://github.com/micadeyeye/webrtc_videochat (2012) Accessed on 3 August 2012.