Albert Abelló Lozano

Performance analysis of WebRTC

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 20.3.2012

Thesis supervisor:

Prof. Jörg Ott

Thesis advisor:

M.Sc. (Tech.) Varun Singh

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Albert Abelló Lozano

Title: Performance analysis of WebRTC

Date: 20.3.2012 Language: English Number of pages:4+13

Department of Communication and Networking

Professorship: Networking Technology Code: S-55

Supervisor: Prof. Jörg Ott

Advisor: M.Sc. (Tech.) Varun Singh

Your abstract in English. Try to keep the abstract short, approximately 100 words should be enough. Abstract explains your research topic, the methods you have used, and the results you obtained.

Keywords: Resistor, Resistance,
Temperature

Preface

Thank you everybody.

Otaniemi, 9.3.2012

Albert Abelló Lozano

Contents

List of Tables

List of Figures

Definitions and abreviations

1 Introduction

The need of a new way to communicate between two points of the planet has been arising as a problem to be solved by our new media distributors. Old systems such as Skype or traditional video calls are not able to cope the needs of the new generations of developers and users that everyday require a more integrated way of communication.

Besides this, the amount of data being transferred during the last years and the prevision for the future allocates a new scenario where non-centralized systems such as P2P are required as data bandwidth grows and systems need to become more scalable. Nowadays networks are still manly content-centric, meaning that data is provided from a source to a client in a triangle scheme, clients upload data to central servers and this data is transferred to the endpoint. This architecture has been provided since long time as reliable and scalable, but with the appearance of powerful applications and media transfer it has been proven to become a real problem.

Those circumstances lead to a whole new world of real-time browser based applications which require also a totally new framework to be developed into. Starting from the online videoconferencing to real-time data applications, for this purpose few attempts where made in the past being highly reliable on specific hardware and custom-built no-compatible system. Taking these decisions made those proposals not be accessible by the massive amount of normal users that could not afford to adapt the requirements.

All the previous concepts are now possible thanks to the increase of performance related to the hardware components available in every average computer nowadays, this increase has helped to build more complex browsers that are able to perform many tasks not just related to web browsing. Having a browser handling OpenGL style of applications is now possible thank to the increase of performance. Besides this multimedia abilities have also been able to reproduce on those browsers and handling webcam media as html is now a reality. Even dough there are still some issues to be considered before being able to freely communicate between two browsers: there is no common standardized protocol that allows developers to do this. WebRTC goal is to approach this problem to build a simple and standard solution for peer-to-peer browser communication [1].

Internet bandwidth has helped to take the decision to start integrating peer-to-peer solutions in browsed based applications, this is due the year-by-year increase of user bandwidth connectivity during the last 10 years. As before the latency was too high being unable to maintain real-time applications working resiliently. But recently the amount of users being able to transfer at high speed has been rapidly increasing as you can see in Figure 1, about 39% of users are now able to download at speeds greater than 4Mbps being this a very good average speed for media content [2].

Regarding the specs on the client side, recent surveys and statistics taken by the game manufacturer Steam [3] has proven that nowadays more than 61% of machines are carrying 1 to 4 gigabytes of RAM and nearly 90% of computers are handling 2 to

	Country	% Above 4 Mbps	QoQ Change	YoY Change
–	Global	39%	-2.8%	6.9%
1	South Korea	84%	-2.2%	28%
2	Switzerland	79%	2.3%	14%
3	Netherlands	79%	-5.0%	-2.0%
4	Japan	74%	2.4%	20%
5	Belgium	69%	-4.7%	-6.3%
6	Canada	69%	0.8%	8.2%
7	Hong Kong	68%	-5.2%	-4.7%
8	Latvia	67%	-1.7%	-2.8%
9	Denmark	66%	5.2%	5.0%
10	Czech Republic	64%	-6.4%	-10%
...				
13	United States	57%	-4.6%	7.9%

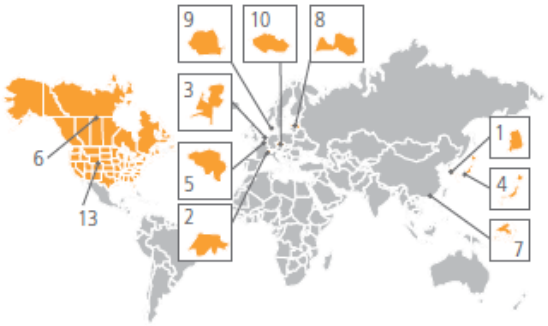

Figure 1: Broadband connectivity statistics about the speeds over 4Mbps around the globe.

4 core CPU with a 64 bit OS, being this environment optimistic for media enhanced applications which require high performance for video encoding and similar. Due to the layering needed to standardize a system like WebRTC running in top of an underlying application such as the browser that handles many processes is very important to rely on a powerful machine. Usually this was not able to be done in the past and one of the challenges has always been performance. They key to success is always to optimize the performance of a process so it does not affect the user experience, one of the challenges of WebRTC.

Traditionally, this concept of performance was approached by the usage of plug-ins or other separate software components which made the system run smoother by avoiding one layer of processing (browser) but being non-standard and not cross-compatible, one of the most import ant concepts when designing applications nowadays. Now the traditional approach has become outdated with the arrival of the new HTML5 where WebRTC is integrated as one of the new APIs available alongside other many different interesting approaches.

1.1 History

Web Real-Time Communication is an API definition with the aim to start an interoperability standard that will help to build P2P applications in the developer layer. The first announcement concept went public in a working group of the World Wide Web Consortium (W3C) in May 2011 [4] and starting the mailing list in April 2011 [5]. During the first stage of the working group the main goal was to define a public draft for the API implementation and a route timeline with the goal to standardize the protocol by ends of 2012. The first public draft of the W3C came public the 27th of October 2011 written by Adam Bergkvist (Ericsson), Daniel C. Burnett (Voxeo), Cullen Jennings (Cisco) and Anant Narayanan (Mozilla) [6]. During this first W3C draft only media (audio and video) where considered as candidates to be sent over the network to other peers, manly focusing in the way browsers will be able to access the media devices without using any plugin or external software.

Alongside to the W3C working group the WebRTC concept also joined the IETF with a working group in May 2011 [7] with the first public announcement charter done the 3th of May 2011 by Magnus Westerlund (Ericsson), Cullen Jennings (Cisco) and Ted Hardie (Ericsson). The milestones of the working group charter initially marked December of 2011 to provide the information and elements required to the W3C for the API design input. On the other side, the main goals of the working group covered the definition of the communication model, session management, security, NAT traversal solution, media formats, codec agreement and transport of the data [8]. Those goals have been evolving during the standardization process and the work done along with the W3C working group.

An important point during the process of standardization came the 1st of June 2011 when Google publicly released the source code of their API implementation [9].

During all this period both working groups have been working alongside to provide a reliable solution to enable applications to perform media and data transfer in a plugin-free environment.

1.2 Support

The following companies have publicly supported and are actively working in the development of WebRTC standards in the W3C: Google, Mozilla and Opera [10]. Other companies such as Microsoft have been publicly supporting a browser-to-browser solution but have provided their own proposal which differs with the one published in the WebRTC group called CU-RTC-Web [11], this proposal did not get much traction by the workgroup being declined to unify with the current specs, during an W3C workgroup poll in September 2012 the chairs of the group decided to attach the development to the already existing WebRTC API instead of moving it to the CU-RTC-Web [12].

During the firsts attempts to build a reliable solution for WebRTC Ericsson Labs publicly presented an initial API based on the preliminary work done in the Web Hypertext Application Technology Working Group (WHATWG), this API was called ConnectionPeer API and required an special module to be installed in your browser [13]. Ericsson lately dropped from the production of it's own browser to focus in the standardization effort and codec discussion leaving the API implementation to the Mozilla and Chrome teams. The original API evolved rapidly during the next months thanks to the workgroup and the developer community feedback thanks to the encourage of the Chrome team to experiment with the unstable API.

1.3 Milestones

During the process of standardization in WebRTC some important moments should be remarked. In January 2012 Opera implemented the first version of WebRTC getUserMedia for accessing the camera and audio [14], during this year getUserMedia is available in the stable version.

Google Chrome integrated the first version of WebRTC in its dev and canary channels of the browser during January 2012 [15], in June 2012 it started moving its API to the stable channel hidden behind a flag, in November 2012 WebRTC becomes fully available in Google Chrome stable channel and is open for user usage [16].

Mozilla Firefox started working on the getUserMedia implementation early 2012 delivering the first version of media access trough API at the beginning of 2012 in the alpha version [17], in April 2012 Mozilla published a WebRTC video demo running on Firefox in the "adler" channel [18], also supporting some primitive DataChannel API. Later in October Firefox Nightly was carrying the first unstable version of the WebRTC API including DataChannel [19], Mozilla announced in September 2012 that the stable version of WebRTC will be shipped along with Firefox 18 in January 2013 [20].

Some announcements have been done from Microsoft as they are also still working in some implementation into Internet Explorer by using CU-RTC-Web as the default standard, at the moment only the Media API information has been published [21].

In the mobile environment there has been less efforts than in the desktop browsers. In October 2012 Ericsson announced the world's first WebRTC-enabled browser for mobile devices called "Bowser" with support for iOS and Android, this browser is able to handle WebRTC calls using RTCWeb Offer/Answer Protocol (ROAP) which is an old discontinued version of the WebRTC API that has moved to Javascript Session Establishment Protocol (JSEP). This browser also differs from the previous desktop alternatives on the codec side, it is carrying H.264 for video and G.711 for audio [22]. The provided API for Bowser is not fully W3C compliant yet.

1.4 Issues in WebRTC

WebRTC uses a mixture of different technologies to perform peer-to-peer communication between clients, those technologies range from SRTP, RTP, RTCP (in the future) and multiple codecs that are being discussed. This scenario makes performance the key point for success in developing stable WebRTC applications in the near future.

Performance is manly related to computer capabilities and the ability to encode/decode at the same time as transferring and monitoring multiple peer connections. All those tasks are now run over the browser not directly on the OS, this is good for interoperability between platforms but bad in the performance aspect.

Media applications are delay sensitive and require a low packet loss for its proper function, WebRTC is working on this trying to implement congestion control over the connection stablished between peers, this work has not been completed yet and will arise as a problem in the near future. Packet loss due to system capacity and bandwidth should be measurable and adaptive in an infrastructure such as WebRTC.

Constraints and bandwidth statistics will make a big difference in how media is performed in WebRTC. Browsers and web applications have been always able to tolerate some amount of delay and packet losses but this is not possible in media infrastructures for real time conversations, a change of scope is needed to be able to

handle Quality of Service (QoS) in WebRTC.

1.5 Contribution

Investigate how WebRTC performs in an application environment trying to evaluate the best way to set multiple peer connections that are able to transfer media in a mesh network. Measure the congestion and statistics of WebRTC in a real environment trying to identify bottlenecks related to encoding/decoding, media establishment or connection maintenance. All this should be able to be performed in real-time over a browser by using the already existing tools or delivering new proposals to the workgroups to increase the QoS in WebRTC.

Using metrics related to RTT, latency, packet loss and bandwidth usage we expect to improve the way WebRTC performs when handling multiple connections.

1.6 Goals

Not sure about here

1.7 Structure

Not sure about here

2 Documentation and drafts

Blah blah blah talk about the sdtrz process and APIs maybe? JSEP vs ROAP? CU-RTC-Web vs WebRTC? VP8 vs H.264 vs Optus? possible stuff to talk that might be interesting to introduce. Short and fast.

2.1 ROAP and JSEP

Since the beginning of WebRTC many proposals have been drafted to accomplish a flexible and usable API for developers. Two different approaches have been designed, in October 2011 RTCWeb Offer/Answer Protocol (ROAP) [23] was introduced being replaced lately by the Javascript Session Establishment Protocol (JSEP) in march 2012 [24]. Both designs are meant to handle the signaling process in the browser simplifying the work for the applications developer, at the same time they should be able to be flexible and adaptable for all the use cases of WebRTC.

During the peer negotiation of the clients SDP is used to understand and set the conditions to transmit the media, this scenario only focuses in media transport between peers, this SDP should be RFC 4566 compliant once the standardization process of WebRTC ends and products are finally delivered. The decision to use SDP for the media negotiations is oriented to make WebRTC capable of SIP/ICE negotiations at the same time of using an understandable and widely used format [1].

At the same time, SDP makes codec negotiation to be standard even using different codec engines, having this a wide usage in environments that require private or licensed codecs.

2.1.1 ROAP

ROAP was initially designed to allow WebRTC browsers to handle the information received form the other peer. This information contains the SDP messages with the agreed information that allows media to be sent.

In ROAP the implementation effort is minimal, this is done by building complex SDP messages that carry all the information, this means that the ICE candidates for the NAT transversal are also packed in the SDP in a JSON format. Considering an example, if a browser A is wishing to establish a media session with browser B, there is some information that must be forwarded from A to B at high level. First, A is willing to start a new session not update an already existing one, it also sends an SDP OFFER that includes media and ICE candidates. If B accepts those parameters and is willing to start the media session it will send back an ANSWER message containing similar information. To confirm that the information is correct A sends an OK message to B, then RTP flow starts. All those messages are related by using a sequence field on the SDP, this helps the browser to maintain and relate all messages when handling more than one peer connection [23].

Even being fully capable to handle media flows this design is not flexible and cannot be adapted to all use cases this reason led to the actual API called JSEP.

2.1.2 JSEP

The ROAP proposal attempted to resolve the problems arising from the control and media plane but had a lack of flexibility for legacy and interaction problems with different protocols, for example, some Jingle designs require ICE transversal process to be started before signaling has finished, this was not possible in the previous approach. Or, being able to renegotiate those ICE candidates in Jingle structures once the initial offer has been sent. This past mechanism is inflexible because it embeds a signaling state machine within the browser, and since the browser itself generates the SDP and controls the possible states of the signaling machine modifying the session descriptions or using alternative states becomes difficult for the developer.

To resolve this flexibility issues JSEP was designed by the IETF [24], this proposal pulls the signaling state machine out of the browser in the Javascript layer. The signaling flow is not anymore automated and the implementation is the part that decides how to handle the messages. Now, there are two differentiated tools required for the API, a way to pass the session descriptions that are negotiated and how to interact and handle the ICE state machine.

Whenever an offer/answer exchange is needed two functions are called createOffer and createAnswer on the PeerConection, those session descriptors are then forwarded into the setRemoteDescription and setLocalDescription functions depending on the side. Those messages are forwarded from both sides using an external technology, WebSockets, polling or similar.

Regarding ICE, with this approach the ICE state machine is decoupled from the signaling part and candidates are not anymore integrated into the SDP body as in ROAP. This method allows ICE to be renegotiated when required and allows the protocol to be friendly with foreigner technologies that also use ICE. When handling those candidates they could also be integrated into the SDP message if required, some protocols like SIP do not decouple the messages, in that case some SDP modification must be done.

This new approach has some drawbacks such as a higher coding level is required, to solve this some libraries that cover most of the negotiation part have been released by developers in order to make it easier for early adopters to use WebRTC. From the other side being able to to send the transport information separately allows the browser to set a faster ICE and DTLS startup reducing the time required for session establishment.

Following the ongoing discussions it can be deducted that JSEP will be the API that will remain and probably will be standardized in the IETF and W3C workgroups.

2.2 The codec war

2.3 CU-RTC-Web vs WebRTC

In August 2012 Microsoft introduced his vision of a real-time communication between browser trying to cover all the WebRTC use cases with a different design [11], this draft collided directly with the ongoing WebRTC proposal done by the W3C

working group [4]. W3C working group decided to attach to the already existing draft some aspects from the Microsoft proposal should be analyzed in comparison with WebRTC. Three main aspects differ from WebRTC:

- No PeerConnection object
- No SDP description or JSEP
- No mantatory codec to be implemented

The ongoing WebRTC proposal identifies a Javascript object called RTCPeerConnection that handles and maintain all the peer connection data transfer, this object handles all the ICE, SDP creation, negotiation and transfer. In the CU-RTC-Web this concept is replaced with a proposal called RealtimeTransport interface that relies in a RemoteRealtimePort interface [25]. This design is a low-level API that forces web developers to build their own application-specific JavaScript code so it can be adapted to every different use case required. No API defined object like in WebRTC, more flexible but much more complicated to implement for application developers.

From the codec perspective CU-RTC-Web approach could be more sensitive to the codec war taking part on the workgroups, one of the key issues in the WebRTC standardization, not setting any codec to be mandatory makes sense considering that after long time there is no consensus about this position yet. On the other hand, it makes sense to set a mandatory codec for a media standard as it will force all the providers to set the same codec and avoid compatibility issues. Codecs being proposed are: VP8, H.264, Optus and G.711.

WebRTC draft RTCPeerConnection relies on a new Javascript spec called Javascript Session Establishment Protcool (JSEP) which help developers to handle the low-layer communication and negotiation tasks [24]. JSEP relies directly on the Session Description Protocol (SDP) [26]. CU-RTC-Web leaves freedom to developers to implement their own communication tasks for their specific application.

The conclusion shows that meanwhile WebRTC moves a lot of work to be done by the browser itself CU-RTC-Web leaves complete freedom to developers to adapt the use case to the proposal instead of adapting the application to the API such as in WebRTC. Both approaches can be valid, but WebRTC makes more sense form the developer point-of-view and it makes it easier to build applications on top of it.

3 Conclusion

The end.

References

[1] H. Alvestrand. Overview: Real Time Protocols for Brower-based applications. Technical report, IETF, 2012. Also available in http://tools.ietf.org/html/draft-ietf-rtcweb-overview-04.

[2] Akamai. The State of the Internet, 2ND Quarter, 2012 Report, 2012.

[3] Steam. Steam Hardware and Software Survey, October 2012. Also available in http://store.steampowered.com/hwsurvey.

[4] Web Real-Time Communications Working Group, May 2011. Also available in http://www.w3.org/2011/04/webrtc/.

[5] Harald Alvestrand. Welcome to the list!, April 2011. Also available in http://lists.w3.org/Archives/Public/public-webrtc/2011Apr/0001.html.

[6] Cullen Jennings Adam Bergkvist, Daniel C. Burnett and Anant Narayanan. WebRTC 1.0: Real-time Communication Between Browsers. Technical report, W3C, October 2011. Also available in http://www.w3.org/TR/2011/WD-webrtc-20111027/.

[7] Real-Time Communication in WEB-browsers, May 2011. Also available in http://tools.ietf.org/wg/rtcweb/.

[8] Cullen Jennings Magnus Westerlund and Ted Hardie. Real-Time Communication in WEB-browsers charter. Technical report, IETF, May 2011. Also available in http://tools.ietf.org/wg/rtcweb/charters?item=charter-rtcweb-2011-05-03.txt.

[9] Google release of WebRTC source code, June 2011. Also available in http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html.

[10] Rian Liebenberg and Jan Linden. Introducing WebRTC an open real-time communications project, April 2011. Also available in http://lists.w3.org/Archives/Public/public-webrtc/2011Apr/0001.html.

[11] Bernard Adoba and Martin Thomson. Customizable, Ubiquitous Real Time Communication over the Web (cu-rtc-web). Technical report, Microsoft, August 2012. Also available in http://html5labs.interoperabilitybridges.com/cu-rtc-web/cu-rtc-web.htm.

[12] Poll for preferred API alternative, September 2012. Also available in http://lists.w3.org/Archives/Public/public-webrtc/2012Sep/0010.html.

[13] Stefan Hakansson. Beyond HTML5: Peer-to-Peer Conversational Video, January 2011. Also available in https://labs.ericsson.com/developer-community/blog/beyond-html5-peer-peer-conversational-video.

[14] Bruce Lawson. getUserMedia: accessing the camera and privacy UI, January 2012. Also available in http://dev.opera.com/articles/view/getusermedia-access-camera-privacy-ui/.

[15] Niklas Enbom. Real-time Communications in Chrome, January 2012. Also available in http://blog.chromium.org/2012/01/real-time-communications-in-chrome.html.

[16] Serge Lachapelle. See you on the web!, November 2012. Also available in https://sites.google.com/site/webrtc/blog/seeyouontheweb.

[17] Robert O'Callahan. MediaStreams Processing Demos, January 2012. Also available in http://robert.ocallahan.org/2012/01/mediastreams-processing-demos.html.

[18] Anant Narayanan. WebRTC efforts underway at Mozilla!, April 2012. Also available in https://hacks.mozilla.org/2012/04/webrtc-efforts-underway-at-mozilla/.

[19] Randell Jesup Anant Narayanan, Maire Reavy and Rob Hawkes. Progress update on WebRTC for Firefox on desktop, November 2012. Also available in https://hacks.mozilla.org/2012/11/progress-update-on-webrtc-for-firefox-on-desktop/.

[20] Robert Nyman. Full WebRTC support is soon coming to a web browser near you!, September 2012. Also available in https://hacks.mozilla.org/2012/09/full-webrtc-support-is-soon-coming-to-a-web-browser-near-you/.

[21] Media Capture API, March 2012. Also available in http://html5labs.interoperabilitybridges.com/prototypes/media-capture-api-(2nd-updated)/media-capture-api-(2nd-update)/info.

[22] Stefan Alund. Bowser - The World First WebRTC-Enabled Mobile Browser, October 2012. Also available in https://labs.ericsson.com/blog/bowser-the-world-s-first-webrtc-enabled-mobile-browser.

[23] J. Rosenberg and C. Jennings. RTCWeb Offer/Answer Protocol (ROAP). Technical report, IETF, October 2011. Also available in http://tools.ietf.org/html/draft-jennings-rtcweb-signaling-00.

[24] J. Uberti and C. Jennings. Javascript Session Establishment Protocol. Technical report, IETF, October 2012. Also available in http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-02.

[25] Bernard Adoba and Martin Thomson. Real-Time Media API. Technical report, Microsoft, August 2012. Also available in http://lists.w3.org/Archives/Public/public-webrtc/2012Aug/att-0063/realtime-media.html.

[26] J. Rosenberg and H. Schulzrinne. An Offer/Answer Model with the Session Description Protocol (SDP). Technical report, IETF, June 2012. Also available in `http://tools.ietf.org/html/rfc3264`.