Live Streaming of Audio and Video through the Browser in an Enterprise

Bergen University College and University of Bergen

Joint Master's Programme in Software Engineering

Håvard A. Heggheim

Supervisors: Lars Petter Helland
Anders Øvreseth

March 13, 2014

Abstract

Media streaming services generate a large portion of the Internet traffic every day with the biggest one being video on demand services. This thesis will focus on the use of streaming media in a collaborative environment within an enterprise setting. There are currently many such services existing today. A lot of these services are costly and use old legacy systems and are therefore not easy to upgrade to adopt the benefits of the latest technologies.

The aim of this thesis is to find the most suitable way for enterprises to implement live media sharing in the web browser. The goal is to detect different problems. In particular this thesis will look into the difficulties of streaming live audio with high quality over the different protocols.

The HTML5 standard introduces many new APIs that give web browsers the ability to communicate directly with each other in real-time. But there are many solutions and the different providers doesn't seem to agree on a single solution. This thesis will examine using these new APIs to see how we can integrate the browser in current media collaboration systems. The goal being to determine the feasability of using these new APIs and evaluate their usage. This thesis hope to aid in developing solutions for media sharing in the browser that can be used in an enterprise setting.

Contents

1 Introduction **6**

2 Background Research **7**

3 Industrial Case **8**

4 Problem Description **9**
4.1 Problem statement 9
4.2 Research question 10
4.3 Research method 10
4.4 Existing work 11
4.5 Expected results 11

5 WebRTC and Virtual Arena **13**
5.0.1 How does Virtual Arena work? 13
5.0.2 How does WebRTC work? 14
5.0.3 What are the differences? 15
5.0.4 What do we need? 16

6 The Solution **18**
6.1 How can we integrate RTC in the Web with Virtual Arena? . 18
6.1.1 The Gateway Solution 18

7 What is the current state of support of Real-time Communication Between Browsers **21**

8 How well does Real-time Communcation scale on mobile devices? **22**

9 What will happen with the support of Real-time Communications in the near future? **23**

10 HTML5 **24**

11 The Approach(experiments) **25**

12 The Implementation(how implemented approach) **26**

13 Evaluation **27**

14 Conclusion **28**

A Appendix Title **29**

List of Figures

Acronyms

Chapter 1

Introduction

Chapter 2

Background Research

Chapter 3

Industrial Case

Chapter 4

Problem Description

As the use of mobile devices increase in business use, Visual Solutions wants to explore the possibility of developing collaboration tools that run directly in the browser using the new API[1]'s drafted by the W3C[2]. This would allow their solutions to extend to more platforms. This chapter aims to explain the problems and what can be done to overcome them.

As described in the previous chapter 3. Visual Solutions has chosen to look at the new Web API's that are under development for doing collaboration services. While there long have been Flash, and other services like Microsoft Silverlight's Smooth Streaming and Apple's HTTP[3] Live Streaming. WebRTC[4] seems to have some important advantages over the other technologies.

- Universal mobile support
- Easy integration, with a high focus on security.
- No need to download clients or plugins.

By looking over the drafts done by W3C, WebRTC seems to be well suited for doing collaboration directly in the web browser.

4.1 Problem statement

WebRTC seems to be the best solution for doing video conferencing on the web, and both desktop and mobile web browsers have some degree of support which improves on a nightly basis.

[1] Application Programming Interface: Specifies how some software components should interact with each other

[2] World Wide Web Consortium:

[3] Hypertext Transfer Protocol:

[4] Web Real-Time Communication: http://www.w3.org/TR/webrtc/

While other solutions like Flash may have better support at the moment, there is a clear indication that the web is moving towards open standards. And a lot of effort is being put into WebRTC at the moment by the big players like Google and Firefox to make this the new standard, however there are still disagreements about which audio/video codecs is going to be used as a standard.

There has been done a lot of functional testing to see if the experiments work. What kind of features and codecs work and what does not. This is done beacuse all the browsers that support WebRTC only support different subsets of the features specified by the W3C.

Find out if WebRTC is a mature and usable technology to be used in a business environment.

Figure out what will happen in the near future with the support for WebRTC on different platforms?

4.2 Research question

This thesis will try to answer the following question:

How can Visual Solutions best integrate WebRTC with their current application Virtual Arena for doing collaboration?

Supporting questions:

- What is the current support for WebRTC in the different browsers?
- How well does WebRTC connections scale with more MediaStreams and peers, and if this can be improved using other models than the standard P2P model
- How will the support for this technology improve in the near future?

4.3 Research method

The research method used in this thesis, is a quantitative experimental method, using systematic investigation and analysis of the different implementations in browsers of the drafted API's by the W3C. Doing experiments to examine the actual workings of the implementations to uncover support for the features.

Practical experiments consists of test cases that was run in Chrome, Firefox, and Chrome for Android. These experiments were performed because there is a lot of incomplete information on the current support.

Many smaller experiments were conducted in the begginning to gain knowledge of the workings of WebRTC. Lots of articles, blogs, books and papers were read.

Looking at trends in the development, this thesis will also be predicting support for the API's in the near future.

4.4 Existing work

There exists a lot of tutorials on creating very simple implementations of WebRTC on the client side, usually setting up one-to-one connections using a third-party service such as Pusher or OpenTOK for doing signaling. More advanced guides shows how you can setup a Node.js server to do signaling using WebSockets.

This is great, but to be able to do what we need, we need to create an advanced server that acts like a client. While most commercial solutions that could potentially be of some help most are closed. But there is one open source project called Licode which utilizes the WebRTCs native libraries and is compatible with the standard and protocols. It's written in C++ and is acts like a MCU[5]. It's open source, but not very well documented. It also doesn't offer what we need in terms of including external RTP[6] streams, but it says in their roadmap that this is planned for the future.

There is no good service on the web that provides live data on how far the support has come on different browsers. There is one service that will check for WebRTC and WebSockets connectivity on http://www.check-connectivity.com/ You will find blogs and articles that provide tables showing who has implemented which APIs, but most will be outdated. The best solution is to test yourself using your own code.

If WebRTC is not supported in a users browser, there may be possible for the user to either update their browser or adjust a flag in the browsers experimental settings.

4.5 Expected results

- From this thesis we want to have provided a solution to how Visual Solutions can integrate WebRTC with Virtual Arena. We want to provide information about all the necessary technologies that need to be implemented.
- We would also like to map the current support for WebRTC on different browsers.

[5]Media Control Unit:

[6]Real Time Protocol: defines a standard packet format for delivering audio and video over IP networks.

- Also we would like to look at how well WebRTC connections scales with more MediaStreams and peers, and if this can be improved using other models than the standard P2P model.

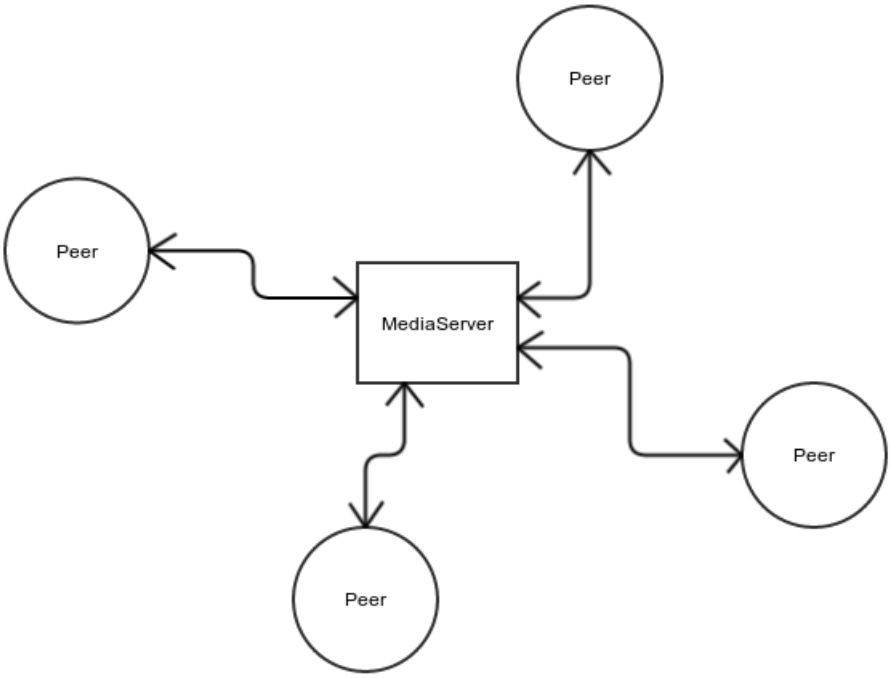
Chapter 5

WebRTC and Virtual Arena

This chapter is about the challenges of making two different media systems work together. It will discuss a possible solution and the pros and cons of this. Then we will look at how Visual Solutions can best integrate this solution and how their vision can be solved.

5.0.1 How does Virtual Arena work?

Virtual Arena supports one-to-one, one-to-many and many-to-many collaborative scenarios. The application uses a MCU that acts as a media server. With this server Virtual Arena can support a lot more incoming and outgoing streams than in a simple peer-to-peer scenario. It also applies mitigation strategies for scenarios with limited bandwidth. Without going into too much detail of how the application is actually put together the main setup looks something like this:



Communication between peers and the media server is done by opening up ports in the firewall to listen for incoming tcp and udp connections. The media server can receives incoming streams and mix it in with the other streams and forward it to all the other peers. All the streams are identified using an SSRC.

Signaling

Virtual Arena uses a proprietary way of doing signaling over RTCP.

Transport

Raw RTP stream over UDP.

Media

Speex for audio and theora for video.

5.0.2 How does WebRTC work?

WebRTC is a very complex synergy of components and protocols. But from a frontend developers point of view, all of this is packaged into three main Javascript API's: getUserMedia, RTCPeerConnection and RTCDataChannel. These are defined by the W3C.

The exchange of real-time media between two browsers follows a process like this:

1 Input devices are opened for capture as the media source. This is done using the getUserMedia API.

2 Now we have to signal the other users that we want to connect to them. using RTCPeerConnection we send an SDP[1] offer to the other clients, which generates an SDP Answer.The SDP here includes ICE[2] candidates. Which opens ports in the firewall. There is a fallback if both clients are on symmetric NAT[3]'s and a connection isn't possible to use a TURN[4] server that acts like a packet mirror, channeling all the packets through the TURN server.

3 Once connection is successful, a DTLS[5] connection is opened and all the media from input devices are encoded into packets and transmitted using SRTP[6]-DTLS streams.

4 At the destination, the packets are decoded and formatted into a MediaStream.

5 The MediaStream is sent to output devices

5.0.3 What are the differences?

While Virtual Arena provides a very simple and not so secure solution media level. This because in a closed enterprised environemnt this is not needed. Most of the security will lie in the firewall anyways.

But since WebRTC is a very open solution and is supposed to work with public users firwall configurations, a lot more complex security architecture is needed.

So while WebRTC packages all their streams in a new format called a MediaStream. At the transport level everything is encrypted using DTLS on top of SRTP.

[1]Session Description Protocol: RFC 4566
[2]Interactive Connectivity Establishment: RFC 5245
[3]Network Address Translator:
[4]Traversal Using Relays around NAT: RFC 5766
[5]Datagram Transport Layer Security: RFC 6347
[6]Secure Real-time Protocol: RFC 3711

5.0.4 What do we need?

There are several problems here. Since Virtual Arena operates on a low-level not much is needed. All we need is to listen to a specific ip-address on a specific port on UDP transmitted data. In the packet headers we will find an SSRC[7] that we use to identify the incoming packets.

With WebRTC on the other hand, we are only allowed to work on a higher level. We cannot access MediaStreams directly without first opening a connection with another peer. This is a problem. This means that we first have to iniate a normal RTC Connection before we can look at any identifiers such as the SSRC. We cannot listen on specific ports for incoming RTC connections, and we cannot specify a specific port to transmit data. Also as of now the current implementation of WebRTC is not designed to add local external streams to the connection. This means that getting WebRTC to work with any external application, we have create some sort of gateway server.

By having done tons of experiments I propose this solution:

One gateway server that can listen to UDP-packets from the Visual Solutions MCU and acts as a peer in WebRTC. This is not an ideal approach, but the most viable at the moment. Ideally one would integrate WebRTC over the whole spectrum, which is both inconveniant and leads to complications.

For a single person that acts as a PeerConnection in WebRTC to listen in on a conversation from Virtual Arena, one would need to initiate a fake RTC Connection using two peers, then create a MediaStream from the incoming UDP packets and inject that MediaStream into the RTC Connection. On the returning side one would have to take the MediaStreams and break them into pure RTC packets with an SSRC identifier in the header and send them in return to the MCU.

First problem is receiving the incoming conversation from Virtual Arena this can be done setting up a socket that listens for incoming UDP packets. This is not a problem and can even be done in a pure Javascript environment using node.js

Then one would have to create a MediaStream Object from these packets. This is not currently possibly using chrome API's or Firefox. In chrome it is possible to create a pure audio MediaStream using the new WebAudio API, and in Firefox it should be possible to create a MediaStream from video using getStreamUntilEnded(), but this is currently broken. In the future however this should be possible using the drafted captureMedia API.

It is however possible to inject external MediaStreams into an RTC Connection.

[7]Secure Real-time Transport Protocol:

For returning data from a PeerConnection to the MCU on would have to record the stream and return the data over a an UDP[8] connection. This should be done using the proposed MediaStream Recording APIs, but none of the browser have currently implemented these yet.

[8]User Datagram Protocol:

Chapter 6

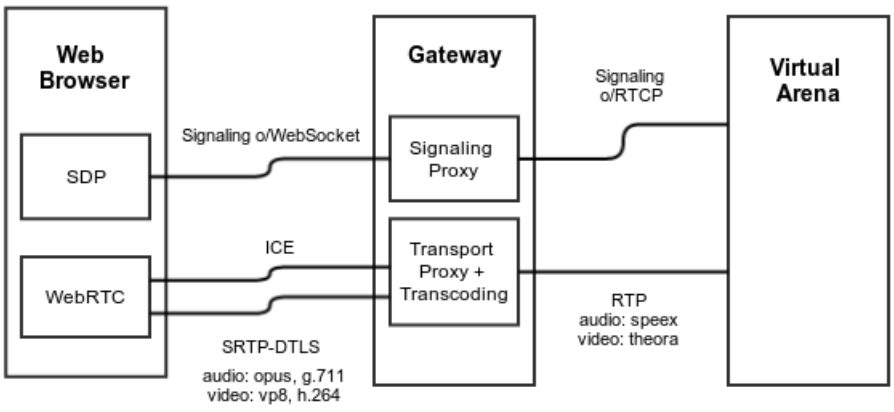
The Solution

6.1 How can we integrate RTC in the Web with Virtual Arena?

There are two solutions to this problem the way I see it. The first is very obvious and complicated, but probably the best solution. The second is more of an experimental hack rather than a viable solution.

6.1.1 The Gateway Solution

The obvious solution is to create a gateway using WebRTC and Visual Arena to turn any WebRTC enabled browser into a client. The gateway will allow the web browser on your preferred device to make and receive connections from Virtual Arena. The gateway would have to contain three modules: a Signaling Proxy, a Transport Proxy, and a Media Transcoder. The global architecture would look something like this:



The Signaling Proxy

Since WebRTC does not define any signaling protocol, one is free to use something custom made. But in this approach, the key information that needs to be exchanged is the SDP, which specifies the necessary transport and media configuration information necessary to establish a connection. This approach is outlined by JSEP[1]. So the role of the Signaling Proxy module would be to extend the custom signaling protocol already used in Virtual Arena to include the necessary metadata provided in the SDP. It would go something lke this:

The MCU sends an offer via the signaling method, then on the client the remote party would install it using the setRemoteDescription() API.

```
...snip...
m=audio 1 RTP/SAVPF 111 103 104 0 8 106 105 13 126
c=IN IP4 0.0.0.0
a=rtcp:1 IN IP4 0.0.0.0
a=ice-ufrag:fAYfQM/iWMQPqiHs
a=ice-pwd:pgbuPPRdpKq+obC0lyRxVDe/
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=rtpmap:111 opus/48000/2
a=maxptime:60
a=ssrc:2209464108 cname:7oIEPieg3XZzHJdN
a=ssrc:2209464108 mslabel:uWu6kVvHhYbbkOtNalf5E2LFgjx4cpGMhnfo
a=ssrc:2209464108 label:2b626a18-c54c-4c1b-9f42-03519a9b63f2
m=video 1 RTP/SAVPF 100 116 117
...snip...
```

The Transport Proxy

The WebRTC specification make support for ICE and SRTP-dtls mandatory. The problem here is that Virtual Arena uses raw RTP streams, it does not need the added security layers that WebRTC defines. It is up to the Transport Proxy to convert the media streams to allow these two worlds to interoperate.

Ice and Security: By modifying a constraint in the IceTransport object we can modify which candidates the ICE engine is allowed to use. We can indicate that the engine must use only relay candidates. This can be used to prevent leakage of IP addresses.

[1] JavaScript Session Establishment Protocol: http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03

The Media Transcoder

The WebRTC specification defines these mandatory codecs:
audio: opus and g.711
video: ?
There are still discussions on the topic of which video codec should be standard. The choice is between VP8 and H.264. The H.264 codec was recently made free by Cisco, so now both choices are royalty free. H.264 is the most widely deployed and currently has the best hardware support, but both Google and Firefox has decided to use VP8 in their WebRTC implementations.

Virtual Arena uses speex for audio and theora for video. So these would have to be transcoded to the appropriate formats.

This is one of the advantages of utilizing a MCU because you can add support for both H.264 and VP8 and be able to create a session between both Chrome, Virtual Arena, and Bowser.

Chapter 7

What is the current state of support of Real-time Communication Between Browsers

Chapter 8

How well does Real-time Communcation scale on mobile devices?

Chapter 9

What will happen with the support of Real-time Communications in the near future?

Chapter 10

HTML5

Chapter 11

The Approach(experiments)

Chapter 12

The Implementation(how implemented approach)

Chapter 13

Evaluation

Chapter 14

Conclusion

summarize your thesis again as in the introduction. Describe how your evaluation revealed that your system is successful. Describe future work in this area.

We have looked at some basic theory about streaming media in the web browser, and explained our experiments and their capabilities. In chapter x we looked at the testing data collected and did an analysis of the results. The results were discussed and conclusions were made. This chapter will summarize the key elements of the results, look at what contributions have been made, and suggest some future work.

To simulate a low latency environment all early experiments were done on a local network. After collecting data using different mechanisms of protocols and streaming media, they were compared with the results from the same experiments using a simulated external network. Data about traffic, latency, and cpu-load were gathered and analyzed. Then we discussed the results with regard to inconsistencies, trends, synergies between different factors, and potential conclusions that could be made.

Contributions to reseach

Future Work Furter testing when different vendors have agreed on a common platform

Appendix A

Appendix Title