Technologies for Real-Time Collaboration using Synchronized Web Browsers in BioInformatics

Brendan D Ball, University of Cape Town

Collaboration is becoming more popular in Bioinformatics and knowledge sharing is the first and easiest way of collaborating. Advancements in technology has given researchers the ability to collaborate despite being in different geo-locations. This collaboration still falls short of real-time synchronization in genome browsers. The success of real-time collaboration in other fields is evidence that it is possible with current technology. This literature review focuses on WebSocket and WebRTC as possible building blocks for real-time synchronized web browsers.

Additional Key Words and Phrases: WebRTC, WebSocket, Real-time collaboration in Bioinformatics, Synchronized web browsers

1. INTRODUCTION

Bioinformatics has become a fast growing field both as a result of advancements in technology and advancements in experimental biological studies. This is evident when looking at the Human Genome Project. These advancements have created a need for collaboration amongst scientists, often working together from across the globe and often across disciplines [Li et al. 2013; Lee 2007; Tsiliki et al. 2014]. One solution to this problem has been the SEQanswers project. This project tries to improve knowledge sharing by means of an open forum where scientists are able to have discussions specific to genomics [Li et al. 2012]. Another successful community-driven project is the NGS WikiBook, which as the name implies, is a wiki containing information on next-generation sequencing and is a big contributor to knowledge sharing in the genomics field. Other platforms exist but are often closed communities and don't contribute as much to knowledge sharing [Li et al. 2013].

Knowledge sharing is just one aspect to collaboration. Other forms of collaboration are also necessary, specifically when researchers are in the process of analysing the results of their experiments, such as sequencing data. There is often a need for synchronized collaboration where researchers discuss their findings while viewing the results. Researchers often use visualisations of the data to analyse it which quickly becomes an unfavourable task when researchers are in different geo-locations. A project called DICODE has been designed for the biomedical field in an effort to solve this synchronous collaboration problem. This project incorporates a discussion view, mind-map view, and formal view (predefined knowledge items) allowing teams to collaborate on a project in different ways [Karacapilidis et al. 2011]. To create a fully collaborative environment there needs to be support for communication, data sharing, and coordination all relying heavily on technical infrastructure [Lee 2007].

This literature review focuses on web technologies which enable collaboration such as communication and data sharing, and specifically synchronized viewing across web browsers. Synchronizing web applications across different web browsers has become a popular feature for enabling collaboration. These web applications are often referred to as groupware. A popular example is Google Docs which is an online text editor. Google Docs enables multiple users to share a single document and are consequently able to view and edit that single document simultaneously. The document synchronizes edits across each user's web browser in real-time [Koren et al. 2013]. More frameworks and libraries are being developed to simplify the integration of synchronization features into any web application [Ozono et al. 2012]. The main problem areas with implementing this synchronization are conflict resolution (if two or more users edit the same area in a document the edits need to be merged) and response times (how long it takes for changes to propagate to other connected users) [Heinrich et al. 2012]. Visualisations can also be synchronized across web browsers, for example viewing a 3D model while moving it around to see it from different angles [Marion and Jomier 2012]. The reason for this literature review is researching the feasibility of implementing synchronization of 2D visualisations, more specifically synchronizing a genome browser to help bioinformaticians analyse a genomic sequence collaboratively. In the background older technologies such as AJAX, Comet, and plugins are briefly described after which the discussion follows, detailing WebSocket and WebRTC which are the two technologies being considered.

2. BACKGROUND

Before we look at new technologies which are most commonly used in groupware, we will look at some older technologies which were used and what their shortcomings are when it comes to implementing groupware. Synchronization requires a bidirectional data stream so that if a user (you) makes a change then that change can be pushed to the other users and if another user makes a change then that change can be pushed to you [Linner 2012].

2.1. AJAX

AJAX stands for asynchronous JavaScript and XML. This technology allows requesting data from a web server using Javascript without reloading the webpage. It is widely used in websites today and it works well. The problem with AJAX is that it is not bidirectional, meaning the web server cannot push data to the client. The client always has to send a request to the web server first and then the web server subsequently responds by sending the data. This limitation can be overcome by polling the web server every interval to receive updated data. Every time a poll is made, a new connection is created with the web server. This is however very inefficient both in terms of the web server's resources and synchronization response times [Gutwin et al. 2011].

2.2. Comet

Comet extends AJAX to overcome its limitation of not being bidirectional. It essentially relies on exploiting AJAX to do things it was never made for. Comet consists of three different hacks which allow bidirectional data streams.

2.2.1. Long polling. sends a request to the web server using AJAX, but instead of receiving a response immediately, the request is only completed when the web server has new data to push to the client. This means the connection between the client and server may be kept open for much longer than a normal AJAX request would. After the client receives data that connection is closed and a new request is initiated. When data is being rapidly updated this becomes as inefficient as normal AJAX would've been.

2.2.2. XHR Multipart Streaming. also uses AJAX but exploits a content type called multipart essentially allowing a connection to stay open between multiple messages. Multipart officially exists to allow large files to be sent in chunks. This often uses up much more memory on the client's device than necessary as it keeps the chunks in memory until the connection is closed. This can however be rectified by periodically closing the connection and creating a new request.

2.2.3. XHR Iframe Streaming. makes use of a hidden iframe which connects to a web server and streams Javascript dynamically. The iframe is then processed to extract the data from the Javascript. This technique also results in memory problems just like multipart streaming above [Gutwin et al. 2011].

These exploits were used when there was no alternative besides using plugins and probably resulted in many unforeseen bugs.

2.3. Plugins

The final alternative to using AJAX or Comet used to be plugins. Plugins are essentially embedded applications into a web page. These applications were written as either Java applets, Flash plugins or Silverlight plugins. This allowed developers to access some native operating system APIs which weren't supported by web browsers themselves. In particular, plugins are able to create TCP socket connections with servers, which allowed access to a bidirectional data stream. This was probably the best solution as it allowed low latency persistent connections without using exploits like Comet. Plugins had other problems though, the first of which being that a client needs to install the plugin before being able to use the web application. Security was also a major problem with plugins, so much so that Apple refused to include Flash support in their iOS browser [Gutwin et al. 2011].

Gutwin and Lippold did an excellent job researching and comparing these technologies using experimental methods. They also go on to compare these technologies with WebSocket and recommend WebSocket to be used rather than Comet or AJAX due to it's better performance [Gutwin et al. 2011]. Plugins have also become infeasible due to security issues and lack of support. [Wenzel et al. 2013]

3. DISCUSSION

Since new web standards have been implemented, namely WebSocket and WebRTC, we no longer have to use sub par solutions to our problem. WebSocket and WebRTC are still new technologies but most updated web browsers support them by now. There are numerous papers exploring the possibilities and performance of WebSocket, as well as papers designing communication architectures on top of WebSocket [Panagiotakis et al. 2013].

3.1. WebSocket

WebSocket is designed to be the TCP socket of the web. Although Http and WebSocket do run on TCP, it is not natively accessible through web browsers. WebSocket was designed to fix this shortcoming and has many advantages over using a TCP socket through a plugin.

WebSocket is a full-duplex communication channel, which means that it allows bidirectional communication asynchronously [Rakhunde 2014]. WebSocket is built on top of existing web infrastructure, which results in both an advantage and a disadvantage [Skvorc et al. 2014]. The advantage being that it is able to handle proxy and firewall traversal which simplifies many problems that plain TCP connections would have given, given that most companies have proxies and firewalls installed [Almasi and Kuma 2013]. The disadvantage is that it is purely client-server and provides no support for peer-to-peer connections, which is a direct result of it being implemented on existing web infrastructure. This will become important when discussing WebRTC later on.

3.1.1. WebSocket protocol. We've already established that WebSocket is built on top of TCP. this means that when a client attempts to initiate a WebSocket connection with the server, it first needs to establish a TCP connection. A successful TCP handshake

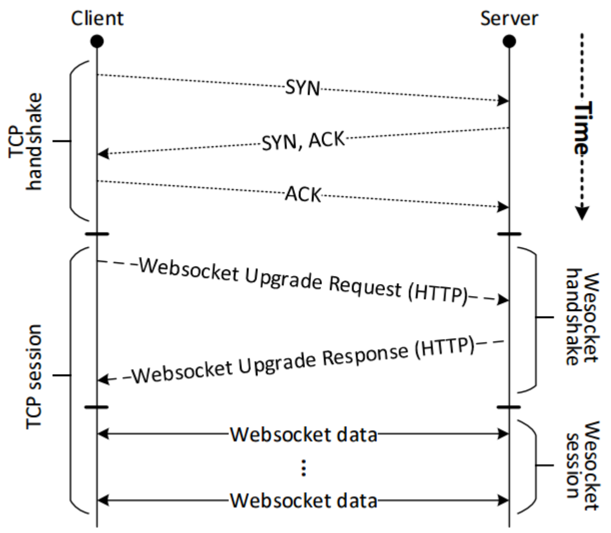

Fig. 1. WebSocket Handshake.

takes three messages. After the TCP handshake is completed, it takes another two messages for a successful WebSocket handshake after which the persistent connection is established and data can be sent back and forth. This protocol is visualized in *figure 1* [Skvorc et al. 2014].

3.1.2. Scalability. When a user requests a web page an http request is sent, a connection is established, the relevant data is transmitted and the connection is closed. This means that http connections were designed to be short lived. This architectural feature benefits the scalability of traditional web applications. WebSocket connections are however persistent and not designed to be short lived. This has a big impact on server resources and makes it more difficult to scale web applications, which use WebSocket, to a large number of users. One possible solution to this is to use distributed clouds on which to host the web servers to improve network traffic and decrease resource utilization for each web server [Solomon et al. 2012].

3.2. WebRTC

WebRTC is a new web technology built on top of RTP (Real-time transport protocol). It's primary goal is to allow real-time audio and video transmission between two browsers without any server relay in between. No plugins are required to use WebRTC as it has been added as a native API to most web browsers. Since it's growth in popularity it has been extended with a data channel as well which allows developers to transmit any arbitrary data [Karadogan 2014].

WebRTC allows a full-duplex communication channel similar to WebSocket. WebRTC is however fundamentally different to WebSocket. WebRTC uses UDP where WebSocket uses TCP. This means that by default WebRTC does not guarantee lossless, ordered data. The advantage of this is that UDP allows lower latency which is better for real-time communication when dealing with audio or video as lossless transmission is not essential. When dealing with a data channel lossless transmission may or may not be essential depending on the use case and WebRTC allows application layer reliable transmission should the developer require it. The last big difference between We-

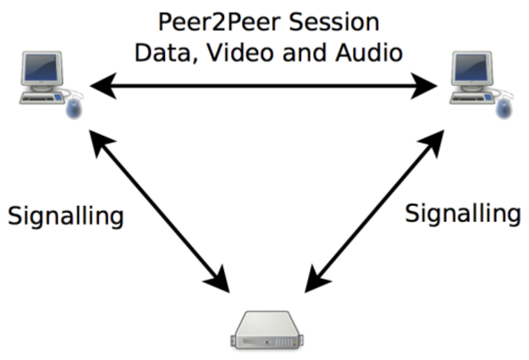

Fig. 2. WebSocket Handshake.

bRTC and WebSocket is that WebRTC is designed to support peer-to-peer (P2P) communication directly between two browsers where WebSocket is entirely client-server [Karadogan 2014]. Vogt, Werner, and Schmidt researched how to take advantage of WebRTC to create a browser based P2P content sharing platform and they were enthusiastic to explore this avenue further [Vogt et al. 2013].

3.2.1. Signalling. P2P connections aren't easy to set up due to complications introduced by NAT (Network Address Translation). The solution to this problem is signalling. Signalling requires a third party, normally a web server to relay connection information between the two clients allowing them to set up a P2P connection. Once the connection is set up, the third party is no longer required and the two clients can communicate while the connection stays open. *figure 2* demonstrates this interaction between the two clients and the third party [Karadogan 2014]. WebRTC requires signalling but imposes no protocol on how it should be done. It is left to the system architects to decide how to implement signalling. There are two common methods used. The first method simply uses AJAX to communicate the necessary signalling information and the second method sets up a WebSocket connection and uses the Session Initiation Protocol (SIP) [Adeyeye et al. 2013].

3.2.2. Scalability. When purely looking at P2P networks they are inherently scalable. However real-time collaboration may very likely exist between more than two users. When trying to synchronize more than two browsers using P2P connections it quickly becomes complex. There are different architectures for solving this problem, the most obvious of which being a full mesh architecture. In a group of size N where the whole group aims to be synchronized, a full mesh requires every peer to set up a connection with every other peer in the group. This results in N^2 connections which quickly becomes infeasible as N grows. An alternative to a full mesh is a star architecture where one client acts as a server in the group and relays data between all clients. This could pose a problem should the main client disconnect but can be solved by implementing a framework that automatically fails over to another client. The last architecture is a multipoint control unit (MCU) which essentially is a client-server architecture where a single server relays data between clients. This architecture would behave similar to a WebSocket architecture [Karadogan 2014].

4. CONCLUSIONS

4.1. WebRTC vs WebSocket

WebRTC has much greater flexibility as it has the ability to function the same way that a WebSocket functions using a client-server architecture in the event that it fails to establish a P2P connection. Another reason for using a client-server architecture could be in a use case where a group aiming to synchronize is too large for a full mesh architecture and benefits more from a single server relaying the data. In cases where full mesh architectures are feasible such as groups with four or less members it is more advantageous to use a P2P network, because it decreases the latency for synchronization and removes resource load off of the main server. In the case of viewing synchronization it is advantageous to use unreliable data transmission with lower latency as this is similar to watching a video stream.

4.2. Further Research

This paper only focuses on comparing WebSocket to WebRTC in the context of real-time collaboration. Further research needs to be done to decide on the signalling protocol for establishing P2P connections using WebRTC. There is also the problem of conflict resolution between synchronized peers. This is a complex problem to solve, especially when dealing with editing in addition to viewing.

REFERENCES

Michael Adeyeye, Ishmeal Makitla, and Thomas Fogwill. 2013. Determining the signalling overhead of two common WebRTC methods: JSON via XMLHttpRequest and SIP over WebSocket. *AFRICON, 2013* (2013), 1–5.

Amir Almasi and Yohanes Kuma. 2013. Evaluation of Websocket communication in enterprise architecture. (2013).

Carl A Gutwin, Michael Lippold, and TC Graham. 2011. Real-time groupware in the browser: testing the performance of web-based networking. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. ACM, 167–176.

Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. 2012. Enriching web applications with collaboration support using dependency injection. In *Web Engineering*. Springer, 473–476.

Nikos Karacapilidis, Manolis Tzagarakis, Spyros Christodoulou, and Georgia Tsiliki. 2011. Facilitating scientific collaboration in data-intensive biomedical settings. In *Biomedical Engineering, 2011 10th International Workshop on*. IEEE, 1–4.

Günay Mert Karadogan. 2014. Evaluating WebSocket and WebRTC in the Context of a Mobile Internet of Things Gateway. (2014).

István Koren, Andreas Guth, and Ralf Klamma. 2013. Shared editing on the web: A classification of developer support libraries. In *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*. IEEE, 468–477.

E Sally Lee. 2007. Facilitating collaborative biomedical research. In *GROUP'07 Doctoral Consortium papers*. ACM, 5.

Jing-Woei Li, Dan Bolser, Magnus Manske, Federico Manuel Giorgi, Nikolay Vyahhi, Björn Usadel, Bernardo J Clavijo, Ting-Fung Chan, Nathalie Wong, Daniel Zerbino, and others. 2013. The NGS WikiBook: a dynamic collaborative online training effort with long-term sustainability. *Briefings in bioinformatics* 14, 5 (2013), 548–555.

Jing-Woei Li, Robert Schmieder, R Matthew Ward, Joann Delenick, Eric C Olivares, and David Mittelman. 2012. SEQanswers: an open access community for collaboratively decoding genomes. *Bioinformatics* 28, 9 (2012), 1272–1273.

David Linner. 2012. *Instant Synchronization of States in Web Hypertext Applications*. Ph.D. Dissertation. Ph. D. thesis, Technical University Berlin.

Charles Marion and Julien Jomier. 2012. Real-time collaborative scientific WebGL visualization with WebSocket. In *Proceedings of the 17th international conference on 3D web technology*. ACM, 47–50.

Tadachika Ozono, Robin ME Swezey, Shun Shiramatsu, Toramatsu Shintani, Ryota Inoue, Yudai Kato, and Takushi Goda. 2012. A real-time collaborative web page editing system WFE-S based on cloud computing environment. In *Advanced Applied Informatics (IIAIAAI), 2012 IIAI International Conference on*. IEEE, 224–229.

S Panagiotakis, K Kapetanakis, and AG Malamos. 2013. Architecture for Real Time Communications over the Web. *International Journal of Web Engineering* 2, 1 (2013), 1–8.

Shruti M Rakhunde. 2014. Real Time Data Communication over Full Duplex Network Using Websocket. (2014), 15–19.

D Skvorc, Matija Horvat, and S Srbljic. 2014. Performance evaluation of Websocket protocol for implementation of full-duplex web streams. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*. IEEE, 1003–1008.

Bogdan Solomon, Dan Ionescu, Cristian Gadea, Stejarel Veres, Marin Litoiu, and Joanna Ng. 2012. Distributed clouds for collaborative applications. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*. IEEE, 218–225.

Georgia Tsiliki, Sophia Kossida, Natalja Friesen, Stefan Rping, Manolis Tzagarakis, and Nikos Karacapilidis. 2014. A Data Mining Based Approach for Collaborative Analysis of Biomedical Data. *International Journal on Artificial Intelligence Tools* 23, 04 (2014), 1460010. http://www.worldscientific.com/doi/abs/10.1142/S0218213014600100

Christian Vogt, Max Jonas Werner, and Thomas C Schmidt. 2013. Leveraging WebRTC for P2P content distribution in web browsers. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 1–2.

Matthias Wenzel, Lutz Gericke, Raja Gumienny, and Christoph Meinel. 2013. Towards cross-platform collaboration-Transferring real-time groupware to the browser. In *Computer Supported Cooperative Work in Design (CSCWD), 2013 IEEE 17th International Conference on*. IEEE, 49–54.