T-111.5360 Report:

Remote Mouse

Valter Kraemer 84669F
Ville Skyttä 42818N
Aalto University

December 26, 2015

1 Introduction

WebSockets is a technology providing an efficient full-duplex bidirectional communications channel between clients and servers. Due to its low latency characteristics, it is well suited for real time applications. The WebSocket protocol is standardized by IETF (Fette and Melnikov, 2011) and W3C (Hickson, 2012) develops an API to enable its use in web pages.

In this report we introduce Remote Mouse, a virtual mouse pointer on a web page, controlled from another web device. WebSockets is used as the communications technology between the devices, through an intermediate server.

[TODO: more]

2 Related work

Bassbouss et al. (2013) address multi-screen web application development and the transformation of traditional web applications to multi-screen capabilities. Both the current and proposed multi-screen application models utilize WebSocket in communications between clients (screens) and servers. [TODO: more about this and how it is related to our stuff]

Agar.io[1] is a massively multiplayer online game. In a nutshell, players control cells in a petri dish, attempting to grow larger by consuming pellets and other cells, and avoiding being consumed by other cells. The game is available for web browsers on its website, and Android and iOS versions are available for mobile devices. The web version uses an HTML canvas and its 2D context for rendering, as well as HTML animation frames. Communication between the browser and the server is implemented using WebSockets. Data is transferred using WebSocket binary frames (WebSocket opcode 2), which are constructed using the ECMAScript 6 ArrayBuffer and DataView objects. Due to the binary nature of the data, the exact semantics of it are not available. During gameplay, the traffic consists of on the order of 50 WebSocket frames per second, with their sizes ranging approximately from a few to 200 bytes.

YouTube has a feature with which it is possible to control another YouTube window's video controls from another screen, such as a computer or a mobile device. It is primarly intended for controlling YouTube on smart TVs, but can also be used in a browser. The controlled screen[2] is operated by its paired remote[3]. The remote works by sending POST requests to the server that forwards them to the controlled window. It also uses polling every 10 seconds to check that the controlled device is still available. The controlled window uses long polling to check if any information is updated. YouTube is using LocalStorage for storing information about the playback device and different identifiers. MediaSource is used to attach sources to their video elements.

Remot.io[4] is a service that controls HTML presentations such as reveal.js[5] from touch based devices. The controlling device sends POST requests to the server that forwards them to the controlled device using long polling by the controlled device. Remot.io is using touch events for their remote. Swipe gestures translate to the directions the user wants to navigate in the slides.

3 Results

[TODO]

[1] http://agar.io/

[2] http://www.youtube.com/tv

[3] http://www.youtube.com/pair

[4] http://remot.io/

[5] http://lab.hakim.se/reveal-js

Latency	Grade	Latency	Grade	Latency	Grade
User 1		User 2		User 3	
500 ms	2	400 ms	2	0 ms	8
200 ms	4	100 ms	5	500 ms	1
100 ms	5	300 ms	4	300 ms	4
0 ms	6	0 ms	8	100 ms	8
100 ms	6	100 ms	8	0 ms	9
0 ms	8	0 ms	8	200 ms	7

Table 1: Subjective user test results

Network	Server	Latency
2G	Heroku	500 ms
3G	Heroku	70 ms
LTE	Heroku	70 ms
Wi-Fi	Heroku	70 ms
Wi-Fi	local	5 ms

Table 2: Typical setup latencies

4 Analysis

[TODO]

To test the subjective effect of latency on user experience, a test with three users was conducted. The users were first asked to use an Apple Magic Trackpad[6] to get a feeling of a local, low latency user experience. Then, they were asked to use Remote Mouse with the latency throttle set to varying settings. The latency settings were shuffled, i.e. not presented in increasing or decreasing order in order to avoid users' expectations affecting the results. Users were tasked to grade the quality of the pointer control experience in scale from 0 to 10, with grade 0 being the lowest one, equal to unusable, and 10 being equally good as the Magic Trackpad.

All three users rated the experience to belong in the middle of the scale at approximately 250 ms latency. 500 ms was classified as barely usable, and 0 to 100 ms quite acceptable.

To aid in estimating how these estimates translate to use of Remote Mouse in different network setups, table 2 lists the typical latencies when the service is running in Heroku and locally, and when it is being used over different network connections.

5 Conclusions

[TODO]

Latency of a network connection is much more important for satisfactory user experience with Remote Mouse than its bandwidth. Bandwidth needs of the application are already quite modest with the current implementation, and could be further reduced, for example by using a more efficient binary WebSocket message payloads, and compression. However, given the already low requirements and possibility of getting negative effects on latency from optimizing for bandwidth usage, these possibilities were not pursued as they are not likely to result in significant overall user experience improvements, if any.

Based on the test conducted as well as the authors' own experiences, the latency goal for acceptable Remote Mouse user experience should be set to the 0 to 100 ms range. According to our test results, these kinds of latencies can be achieved with 3G and better mobile network connections; 2G connectivity is not sufficient.

[6] `https://en.wikipedia.org/wiki/Magic_Trackpad`

References

Louay Bassbouss, Marc Tritschler, Stephan Steglich, Kiyoshi Tanaka and Yuji Miyazaki. Towards a Multi-screen Application Model for the Web. *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, pages 528–533. IEEE, July 2013. doi: 10.1109/COMPSACW.2013.96.

Ian Fette and Alexey Melnikov. The WebSocket Protocol. RFC 6455, December 2011. URL `http://www.ietf.org/rfc/rfc6455.txt`.

Ian Hickson. The WebSocket API. Technical Report, W3C, September 2012. URL `http://www.w3.org/TR/2012/CR-websockets-20120920/`.