A Network Extension for Gamemaker HTML5

Teun Kokke

Undergraduate Dissertation
Software Engineering
School of Informatics
University of Edinburgh

2016

Abstract

summarising the report
Creating an extention for Gamemaker creations, to allow fast networking with maximal reliability.

QUESTIONS TO SUPERVISOR

- Need to mention examples in background for "why gamemaker"? if no, where should examples be given if at all?
- Known from previous experiments: a single instance cannot simulate more than roughly 700 clients without causing losing stability. Do I have to show this in detail with a graph as part of the experiments, or should I just mention that eg. 500 clients per instance is arbitrary for this reason and ignore the details?
- Is it required to include formulae for mean, variance and standard-deviation?
- Found tex templates from university on website. Use those, or keep this one?

TODO

- give definition for server-reponse time
- Add abbreviation / terminology legend in preamble, eg TCP, UDP, RTT, CPU time, CPU usage, RSS.
- in section "fairness depending on location", add world map with tested locations and their average mean
- Consider actual playability in a game OR add assumption how RTT affects atual usage of an application (find reference?)
- fill in further details on experiment setup (text format)
- Write down sections for literature, references
- Write down details about implementation
- Possibly add new implementation features / clean up existing implementations.
- Contact community

Acknowledgements

First of all, I would like to thank my supervisor Dr. Myungjin Lee for guiding me through the process of writing the report and critisizing my work.

I am also grateful to those people across the globe who have assisted me with testing and collecting data for the evalutation experiments.

My sincere gratitude to my family, friends, the Dutch Gamemaker community and the Yoyogames forums, for providing me with feedback and ideas without which this project would not have been the same.

Table of Contents

Chapter 1

Background and Related Work

1.1 Background

In the current day and age, we spend a large portion of our time using web applications. A vast amount of the internet consists of services supported by web applications, and browser games are as popular as ever.

There exist many good reasons for this to be true. Browser applications and games do not require prior installation. They are therefore easy to start up, safe from viruses and don't require an admin-user account in order to be executed[1]. They are able to interact with other web applications. They are highly platform independent and potential users are generally easy to reach. Also their updates are seamless and can be implemented without requiring patch downloads to a harddrive.

1.1.1 Networks in Applications

Humans are naturally social beings, are found to be more drawn into games when this social aspect is being provided, and are therefore more likely to return in the future[2]. When allowed, we can share a sense of community, develop our own social identity within the application, and regularly seek social support from peer users, even about non-related and real-life topics[3]. Users of networked applications are much **more likely to advertise** the application in their social circles than they are for non-networked applications.

1.1.2 Network Fairness

Having that said, one must consider the technical aspect of the network: if the application is in the form of a game, **playability and fairness are crucial for an enjoyable gameplay**. This is especially true for games containing elements where speed or response time is important.

In a typical networked game, game clients are described by a **limited set of parameters** received by a server. These parameters represent the "game state". **When due to delay between the clients and server the game state is desynchronised, fairness is reduced**[4].

1.1.3 TCP and UDP

Choosing the right protocol to handle the data transmissions is therefore crucial. Transmission protocols are one of the contributors that will heavily characterise the network.

Generally speaking, TCP is a form of reliable data transmission but contains therefore extra overhead, making it therefore known to be slightly slower than UDP when the network has little packet loss.

As aforementioned, Websocket is a protocol built on TCP in 2011. However, its sister WebRTC which is built on UDP, is still vaguely "under construction"[5]. No proper standards are out yet, which combined with the fact that UDP requires manual rule control by the developer, makes UDP many times **more complicated** for developers to handle efficiently.

1.1.4 HTML5

1.1.4.0.1 HTML5 is a raising web standard released in 2014 by the World Wide Web Consortium (W3C). It is designed to be cross-platform and runs on most modern web browsers such as Google Chrome, Mozilla Firefox, Apple Safari, and Opera [1]. Also mobile web browsers that come preinstalled on iPhones, iPads and Android phones support HTML5.

It superseeds its predecessors HTML4 and XHTML 1.1 with the aim to reduce the dependence of functionality from third-party plugins such as Flash and Java applets, which are either deprecated or entirely unsupported by most devices [6].

Scripting is replaced in HTML5 by markup where possible, causing the world of browser-gaming to change rapidly. One of the the newly introduced features is the <canvas> element, which is defined as "a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics or other visual images on the fly"[7]. The element can thus be used to draw graphics in JavaScript with the "Canvas API"[8].

1.1.5 Node.js

Node.js is a JavaScript interface with the aim to create "real-time websites with push capability" allowing developers to work in the "non-blocking event-driven I/O paradigm" [9]. This means that developers can use it to create real-time web applications where a

[1]HTML5 supported browsers are found at `https://html5test.com/results/desktop.html`

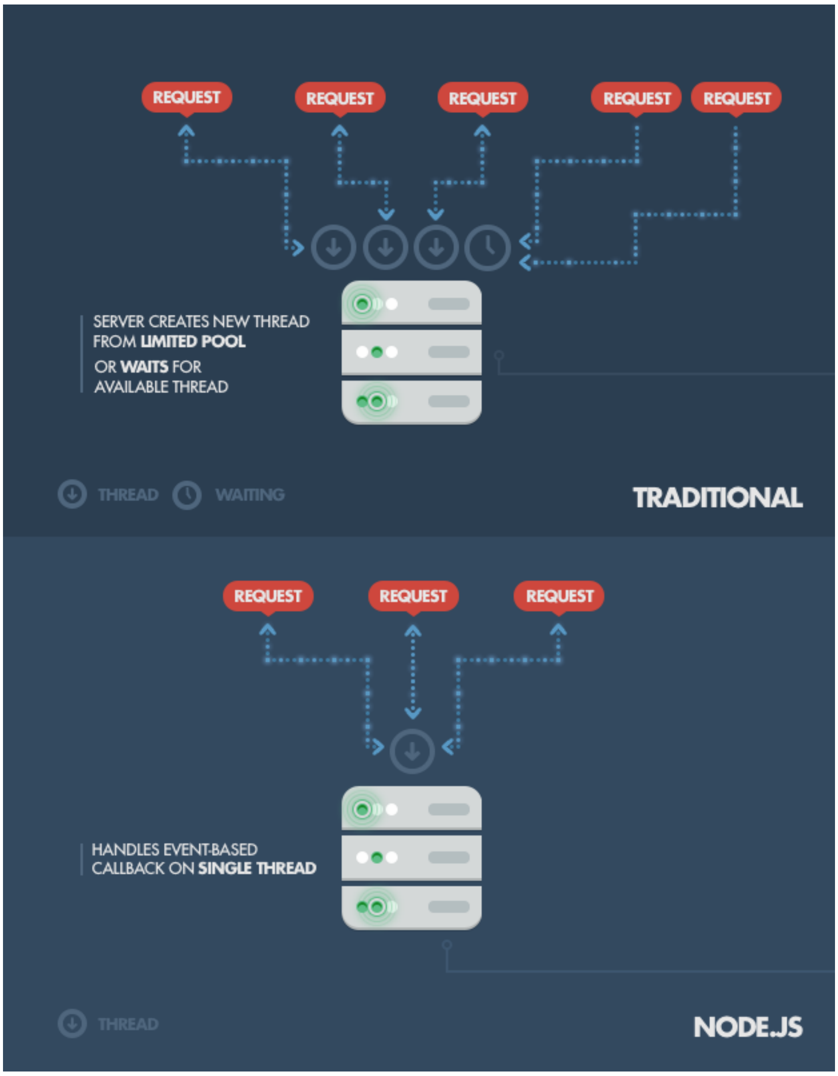

Figure 1.1: image displaying the comparison of clients connected to a Node.js server versus clients connected to a traditional server. The traditional (unscalable) system which would rapidly run out of RAM as it assigns new clients to their own thread[11]

.

server and client can both initiate communication, and both can exchange data freely without repeatedly having to refresh the webpage.

In short, new client connections get allocated to a heap in the memory and client events are handled on a single thread by the server's operating system without choking the Node.js event loop. This therefore allows servers running Node.js to maintain thousands of concurrent connections without running out of RAM memory[10], as opposed to traditional servers which create a new thread for each client.

1.1.6 Socket.io

Socket.io is an event-driven JavaScript library that can be used both on the client's browser, and a server[12]. It is capable of sending and receiving data via WebSockets (handled through a TCP connection) without interruption of the code flow[13][14]. For this reason, it is often used in combination with Node.js.

1.1.7 GameMaker

1. Gamemaker does **not** support networking for HTML5 (js) export
2. The current extentions that exist for gamemaker are limited to a basic use of TCP
3. Webbrowsers do not generally support the use of UDP.
   Suggested methods are developing a java or flash applet, which will handle the networking features.
   A recent technology called *WebRTC* allows js to use UDP data transfer directly on the web-client, but security features within the network make holepunching complicated.

1.1.7.0.1 Gamemaker by YoyoGames is a software creation tool with the aim to simplify and speed up game and app development. It is cheap, simple to learn and flexible to use, making the software demanded by small teams, professionals and novice developers. Consequently, the human nature of staying within trusted environments causes developers to stick to Gamemaker.

During the rise of HTML5 and the growing popularity of Gamemaker, YoyoGames has provided the functionality to export any application to a JavaScript program that can be executed directly in the browser. This way i.a. updates can be made seamless, require no file-download nor installation, is platform-independent and easier to distribute.

Some features are lost during the transition from a windows-executable to a web-application. One of them is a critical component to application popularity: networking functionality. Since the HTML5 update in September 2011, this feature was never added.

The implementation in this Honours project will allow developers to again add networking features to their web-browser applications.

1.2 Related Work

1.2.1 JSiso

http://jsiso.com/

1.2.2 Unity

1.2.3 Unreal Engine 4

1.2.4 CryEngine

1.2.5 Havok Vision Engine

1.2.6 Project Anarchy

1.2.7 ShiVa

1.2.8 App Game Kit

1.2.9 GameSalad

1.2.10 ...

Chapter 2

Literature Review

2.1 Multiplayer Networking in Modern Game Engines

2.2 Fairnesss and Playability in Online Multiplayer Games

2.3 ...

Chapter 3

Development

3.1 Design

3.1.1 Prior Considerations

3.1.2 Server and Client

3.1.2.1 Good Coding Practices

3.1.2.2 Interaction

3.2 Implementation

3.2.1 Server

3.2.2 Client

3.2.3 The Extension

3.3 Applications Developed with the Extension

3.3.1 Benchmark Application

3.3.2 Real Game Application

3.3.3 Developer Template

Chapter 4

Network Extension Evaluation

4.1 Setup

4.1.1 Controlled Network Experiments

The controlled experiments were conducted in order to predict server behaviour when loaded under similar pressure in future occasions.

4.1.1.1 Concurrent Connections Specifics

4.1.1.1.1 Environment

- Variable number of concurrent connections cc, up to 15 instances with each simulating at most 500 clients.
- Clients do not contact the server after establishing a connection.
- The CPU usage, time and RSS (Resident set size) are recorded after each every time another group of 100 clients connect to the server.

4.1.1.1.2 Dependent variables

- CPU usage on the server
- CPU time on the server
- RSS on the server

4.1.1.2 Message Broadcasting Specifics

4.1.1.2.1 Environment

- 5000 concurrent connections (10 instances each simulating at most 500 clients).

- Each package that is sent has a size of 8 bytes.
- Each client sends packages at regular time intervals, causing the server to handle n messages every second.
- Each client measures the roundtrip time of its package to the server.

4.1.1.2.2 Dependent variables

- Mean roundtrip time between all clients
- Variance of the roundtrip time between all clients
- Standard Deviation of the roundtrip time between all clients

4.1.1.3 Server hardware and network specification

4.1.1.3.1 The controlled network experiments were executed on a Windows 7 64-bit platform with 12.6GB available physical memory and an Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz processor. The software included Node v0.12.3 and Socket.io v1.3.7 in order to handle the networking operations.

The network was controlled using Dummynet, using a below-average UK household network setup. This involves an upload speed of 5Mbit/s, and a download speed of 1Mbit/, although no packet loss was set in order to ensure consistency throughout the controlled network experiments.

4.1.2 Real Network Experiments

The network was consistently running with a 54.0Mb/s download and 3.0Mb/s upload speed.

4.1.2.0.1 Multiple tests were executed with clients being located at specified locations. In each case, the clients were sending 8-byte messages to the server at a regular interval of 1 second for 2 minutes. The 120 roundtrip times for each client were then averaged in order to blur occasional peaks. At this point the roundtrip times of the clients in each common location were averaged, and then the deviance of the roundtrip times at each of these locations were calculated.

All network experiments were executed using a single server located in Edinburgh and in each experiment all clients were connected and communicating to that server simultaneously.

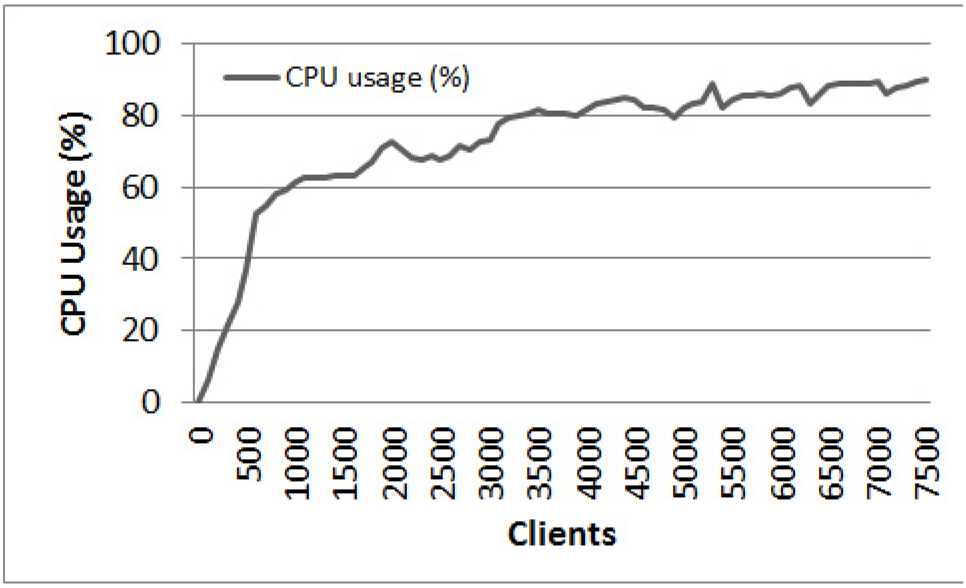

Figure 4.1: Displaying the CPU usage for the server process in percent.

When the CPU usage is above 70 percent, the user may experience lag. Such high CPU usage indicates insufficient processing power. Either the CPU needs to be upgraded, or the user experience reduced.

4.2 Results

4.2.1 Controlled Network Results

4.2.1.1 Concurrent Connections

The following experiment evaluates the server performance with regard to the number of clients.

4.2.1.2 Message Broadcasting Performance

The following experiment evaluates the number of messages that can be handled by the server simultaneously, and considers how this affects the fairness in response-time of the individual clients.

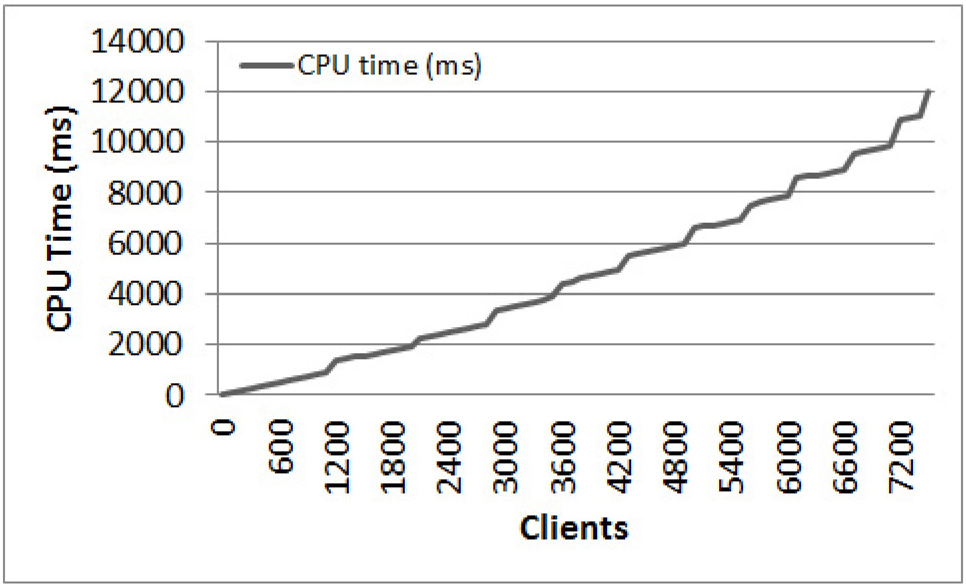

Figure 4.2: CPU time in milliseconds, displaying the amount of time required for the server to process the clients.

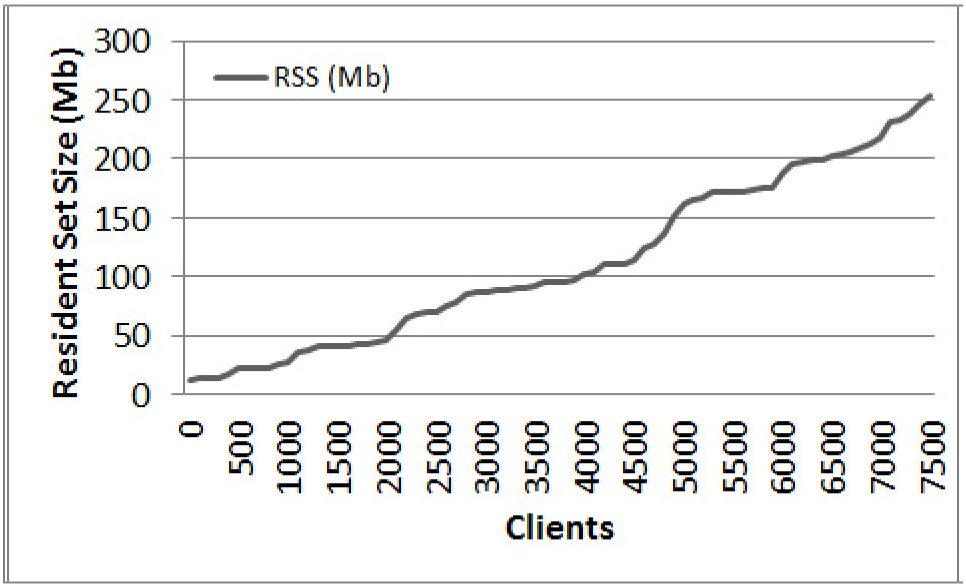

Figure 4.3: Resident set size in Megabit, showing the portion of RAM that is occupied by the server process.

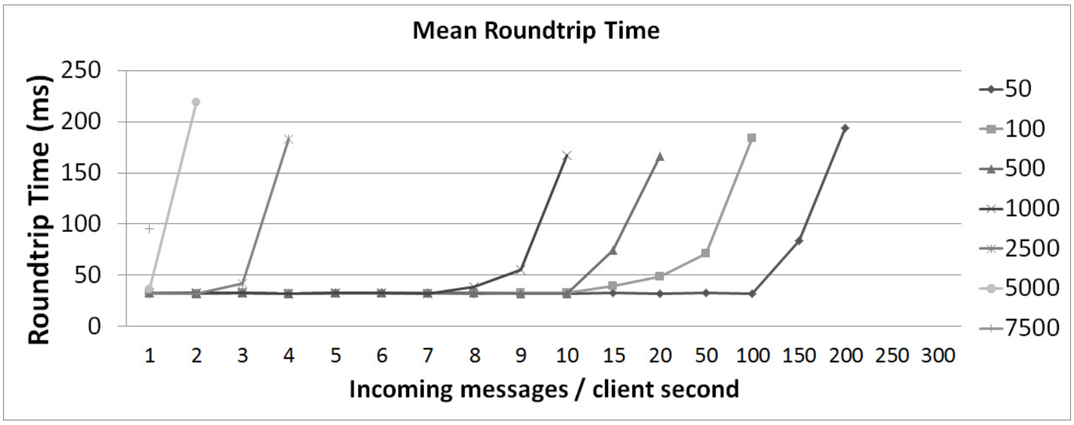

Figure 4.4: The mean roundtrip time of all the messages that pass through the server. As expected, with few concurrent clients connected to the server, the server manages to broadcast many more messages.

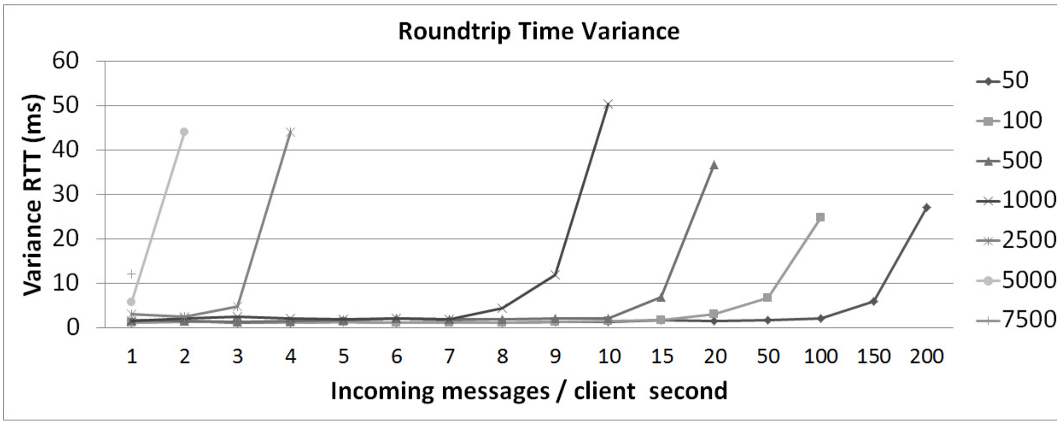

Figure 4.5: The variance of the roundtrip times, showing the fairness between clients decreases significantly when the server receives messages faster than it can broadcast.

4.2.2 Real Network Results

4.2.2.1 Location-wise Delay Fairness

The following experiment evaluates the effect of the geographical distance between groups of clients and the server with respect to fairness in response-time from the server to the clients.

4.2.2.1.1 Test cases:

1. Local network setting: Five clients physically located in the same local home network.
2. Same city: Five clients physically located in Edinburgh (LAN excluded).
3. Same country: Three clients physically located in Scotland: Edinburgh, Glasgow and Dundee.

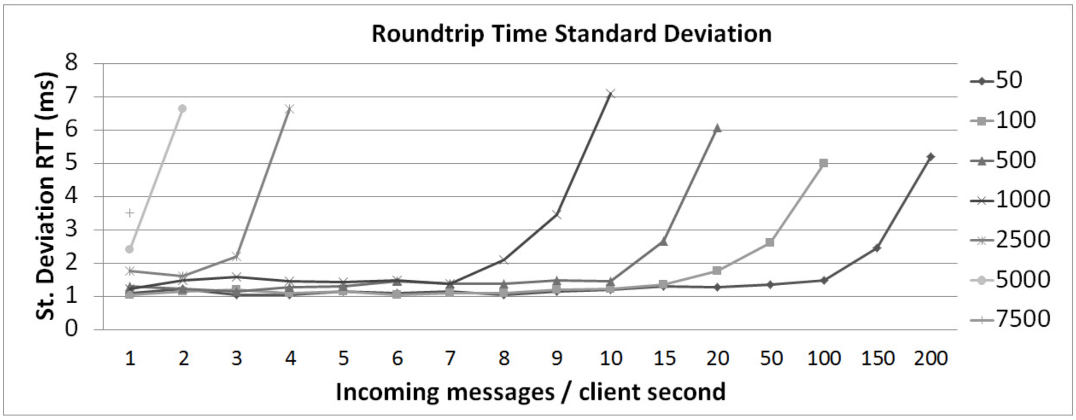

Figure 4.6: Standard deviation of the message roundtrip times.

4. Europe: Five clients physically located in the United Kingdom, Hungary, France, Germany and the Netherlands.
5. Inter-continental: Eight clients physically located in South Africa, California (USA), India, Thailand, Germany, Hungary, United Kingdom and the Netherlands.

4.2.2.1.2 Results: Average roundtrip times per location:

Location	Average RTT
LAN	32ms
Edinburgh	55ms
Dundee	65ms
Germany	67ms
Glasgow	68ms
France	69ms
the Netherlands	70ms
United Kingdom	71ms
Hungary	76ms
India	103ms
South Africa	144ms
California (USA)	158ms
Thailand	171ms

Average roundtrip times per test:

Test case	1	2	3	4	5
Mean RTT	32.1ms	54.8ms	62.7ms	70.6ms	107.5ms
Variance RTT	1.3ms	8.2ms	46.3ms	11.3ms	1900.9ms
Std RTT	0.4ms	2.9ms	6.8ms	3.36ms	43.6ms

Chapter 5

Conclusion and Future Work

5.1 Conclusion

5.1.1 Comparison of the Extended GameMaker Functionality with Related Work

5.1.2 Criticism on the Implementation and Design Decisions

5.2 Future Improvements

[15] [4] [16] [7] [17] [18] [6] [9] [10] [11] [13] [14] [12] [8] [1] [3] [19] [2] [5]

Bibliography

[1] Vinny Lingham. Top 20 reasons why web apps are superior to desktop apps. http://www.vinnylingham.com/top-20-reasons-why-web-apps-are-superior-to-desktop-apps.html, 2007. [Accessed: 2016-01-16].

[2] Christoph Klimmt, Hannah Schmid, and Julia Orthmann. Exploring the enjoyment of playing browser games. *Cyberpsychology & behavior : the impact of the Internet, multimedia and virtual reality on behavior and society*, 12(2):231–234, 2009.

[3] Erin L. O' Connor, Huon Longman, Katherine M. White, and Patricia L. Obst. Sense of community, social identity and social support among players of massively multiplayer online games (mmogs): A qualitative analysis. *Journal of Community & Applied Social Psychology*, 25(6):459–473, 2015.

[4] Jeremy Brun, Farzad Safaei, and Paul Boustead. *Fairness and playability in online multiplayer games*. PhD thesis, University of Wollongong.

[5] Ilya Grigorik. *High Performance Browser Networking*.

[6] Ian Elliot. Death of flash and java. http://i-programmer.info/news/86-browsers/8783-death-of-flash-and-java.html, 14 July 2015. [Accessed: 2016-01-16].

[7] Mark Pilgrim. *HTML5: Up and Running*. O'Reilly Media Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2010.

[8] Mozilla Developer Network. Canvas api. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API, 2015. [Accessed: 2016-01-16].

[9] Tomislav Capan. Why the hell would i use node.js. http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js, 2014. [Accessed: 2016-01-16].

[10] Mike Pennisi. Realtime node.js app: A stress testing story. https://bocoup.com/weblog/node-stress-test-analysis, 2012. [Accessed: 2016-01-16].

[11] Alejandro Hernandez. Init.js: A guide to the why and how of full-stack javascript. http://www.toptal.com/javascript/guide-to-full-stack-javascript-initjs. [Accessed: 2016-01-16].

[12] socket.io. http://socket.io/. Homepage, [Accessed: 2016-01-16].

[13] Drew Harry. Practical socket.io benchmarking. http://drewww.github.io/socket.io-benchmarking/, 2012. [Accessed: 2016-01-16].

[14] Mikito Takada. Performance benchmarking socket.io 0.8.7, 0.7.11 and 0.6.17 and node's native tcp. http://blog.mixu.net/2011/11/22/performance-benchmarking-socket-io-0-8-7-0-7-11-and-0-6-17-and-nodes-native-tcp/, 2011. [Accessed: 2016-01-16].

[15] Alan Edwardes. Multiplayer networking in a modern game engine.

[16] Peter Lubbers, Brian Albers, and Frank Salim. *Pro HTML5 Programming*. Paul Manning, Springer Science and Business Media, LLC., 233 Spring Street, New York, 2010.

[17] Nathaniel S. Borenstein. *Programming as if People Mattered: Friendly Programs, Software Engineering and Other Noble Delusions*. Princeton University Press, Princeton, New Jersey, 3rd edition, 1994.

[18] Mark Overmars. Teaching computer science through game design. *Journal Computer*, 37(4):81–83, April 2004.

[19] L Suarez, C Thio, and S Singh. Why people play massively multiplayer online games? *International Journal of e-Education, e-Business, e-Management and e-Learning*.