A Network Extension for GameMaker HTML5

Teun Kokke
s1242775

Undergraduate Dissertation
Software Engineering
School of Informatics
University of Edinburgh

2016

Abstract

summarising the report

Questions to Dr. Lee

- what to do with the footnotes? make them regular reference? if yes, just the url? or are there specific standards.

COMPLETED

- planning the project
- general research on the topic, ie. answer the question: "where to start?"
- developing the implementation
- develop application using the working extension to test if it works (both locally and online)
- evaluation: assessing the limits of the implementation (given a specific server specification). Both locally and online (different distances to server, different cities? countries? continents?)
- write networking extension implementation
- execute experiments to evaluate the implementation

TODO

February

- give definition for server-reponse time
- 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?)
- write literature review and development section
- transform details of experiment setup in text format (+ write additional details if details are missing)
- Write down details about implementation, design decisions must be well justified according to literature
- Possibly add new implementation features / clean up existing implementations.
- explain the evaluation results

March

- clarify thesis and write intro

- Record video of working implementation

- Create proper template for other developers to use + writing down a guide how to use it, where to start etc.

- Release the project to the community

- Finalize report

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 giving feedback of 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.

List of Acronyms

- API - Application Program Interface
- IP - Internet Protocol
- TCP - Transmission Control Protocol
- UDP - User Datagram Protocol
- W3C - World Wide Web Consortium
- P2P - Peer-to-Peer
- RTT - Roundtrip Time
- RSS - Resident Set Size: the portion of the process's memory held in RAM
- CPU - Central Processing Unit
- RAM - Random Access Memory
- LAN - Local Area Network
- GUI - Graphical User Interface
- IDE - Integrated Development Environment
- IETF - The Internet Engineering Task Force

Table of Contents

Chapter 1

Introduction

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. 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.

The market in this area is therefore booming. Developers all across the world are trying to be the fastest at developing their games, in the easiest way possible. The easier the process, the faster the development. The faster the development, the sooner the game can be released.

One of many developer tools that aims for exactly these two ideals is GameMaker Studio. However due to the feature limitation of creating networked applications, developers may be forced to use less suited software instead.

This paper therefore investigates the quality of related software. It also demonstrates how an extension can be added to GameMaker, one that will add networking features to HTML5 games and applications. Additionally, the server's behaviour will be assessed based on the specified setup, and suggestions will be added based on other literature in order to improve this further.

Chapter 2

Background

2.1 Background

2.1.1 Network Fairness

Different applications often have different demands from the network. One must therefore 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[2].

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**[2].

2.1.2 TCP

TCP is a highly reliable connection-oriented and error-free host-to-host data transmission protocol. After establishing a connection using an event referred to as the "three way handshake" [1], it keeps the connection open [2]. It automatically takes care of handling retransmission of dropped packets and acknowledgement of arrived packets. For this reason it is one of the most common transmission protocols in applications that require reliable data transmissions. One down-side however is that due to the additional error-handling, the packets are relatively large and thus cause some overhead.

[1] http://www.inetdaemon.com/tutorials/internet/tcp/3-way_handshake.shtml
[2] https://tools.ietf.org/html/rfc793

2.1.3 UDP

UDP is a connectionless transmission protocol that, unlike TCP, provides a "minimal, unreliable, best-effort message-passing transport to applications" [3]. It provides no guarantees for delivery and no protection from duplication. Despite this clear downside, due to the small header size it may often be useful for applications that value the speed of the connection above reliability of the content.

2.1.3.1 NAT traversal

Due to the limitation of IPv4 addresses, networks often consist of local networks; groups of devices that are mapped to the same global IP address. Network address translation (NAT) is a mechanism that takes care of assigning IP addresses to local devices within the network, whilst maintaining the same global IP to identify the local network as a whole.

As UDP does not maintain an open connection between the hosts, packets have to pass the NAT repeatedly, and can easily be blocked by the firewall. NAT traversal techniques such as "hole punching" are therefore established, Although they are not applicable in all NAT devices or situations [4].

2.1.4 WebSocket

WebSocket is a protocol that "provides a method to push messages from client to server efficiently and with a simple syntax" [3]. It was standardized by the IETF in December 2011, providing a full-duplex (two-way) communication between client and server through a single TCP connection. The goal is to "provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections" [5]. This is done by setting up a socket, which is essentially a door that leads to a specific host, through which data can be sent.

2.1.5 Socket.io

Socket.io is an event-driven JavaScript library that can be used both on the client's browser, and a server[4]. It supports features that allow sending and receiving data using the WebSocket protocol, without interruption of the code flow[5][6]. For this reason, it is **often used in combination with Node.js**.

[3] http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html
[4] https://tools.ietf.org/html/rfc5128#page-11
[5] https://tools.ietf.org/html/rfc6455

2.1.6 WebRTC

As aforementioned, Websocket is a protocol built on TCP. However, it has a sibling: WebRTC. WebRTC is built on UDP, allowing scalable and fast networking opportunities, but is still vaguely "under construction"[7]. Although certain web applications have already been created with WebRTC (mainly for P2P video streaming [8]), no proper standards are out yet[1].

This, combined with requirement of manually specifying the rules of communication, as well as the fact that WebRTC has to circumvent network security and privacy in order to allow web browsers to transmit data over UDP[8], makes UDP many times **more complicated** for developers to handle efficiently.

2.1.7 HTML5

HTML5 is a raising web standard released in 2014 by the 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 [6]. Also mobile web browsers that come preinstalled on iPhones, iPads and Android phones support HTML5.

It superseeds its predecessors HTML4 and XHTML1.1 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 [9].

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"[10]. **The element can thus be used to draw graphics in JavaScript with the "Canvas API"**[11].

2.1.8 Node.js

Traditionally, servers create a separate thread for each client, therefore rapidly running out of RAM and keeping clients on hold until memory for a new thread is released[12].

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" [12]. This means that developers can use it to create real-time web applications where a server and client can both initiate communication, and that 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

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

the (Node.js) event loop. **This therefore allows servers running Node.js to maintain thousands of concurrent connections without running out of RAM memory**[13][14], as opposed to the traditional, less scalable servers.

2.1.9 GameMaker

GameMaker by YoYoGames is a software creation tool with the aim to simplify and speed up game and application development. There have been several hits on the market for games developed with Gamemaker such as "Reflections", "Rick O'Shea" and "Simply Solitaire" [15].

Developing applications and games in GameMaker is cheap, simple to learn and flexible to use, making the software demanded by small teams, professionals and novice developers [16]. Sandy Duncan, the founder and former chief executive officer of YoYoGames stated in a phone interview that they have never lost money on a game that they developed with their technology[15].

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[17].

Some of the features that are normally supported by GameMaker are however lost during the transition to a web application. One of these features is the networking functionality. Thus far since the update to export GameMaker applications to HTML5 in September 2011, YoYoGames has never included this feature[7]. An attempt was made by a member of the YoYoGames community, however it is considered to be in alpha stage and fails to be used by other developers[8]. It is a basic setup that merely broadcasts messages as they come in.

The main GameMaker community forum is hosted directly by YoYoGames[9], serving 280,000 registered users recorded in January 2016, with a combined total of 220,000 topics and 3,480,000 posts. There also exists a notable GameMaker forum in the Netherlands[10] which hosts an additional 557,000 posts for 56,000 topics for almost 18,000 members.

2.2 Fairnesss and Playability in Online Multiplayer Games

more detail on Network Fairness, causes and fixes to unfairness

[7] http://docs.yoyogames.com/source/dadiospice/002_reference/networking/index.html
[8] https://github.com/amorri40/39js
[9] http://gmc.yoyogames.com/
[10] http://www.game-maker.nl/forums/

2.3 Multiplayer Networking in Modern Game Engines

game state maintained through a limited set of parameters

Chapter 3

Related Work

Being a cross-platform game development tool primarily for 2D graphically oriented games, GameMaker has many competitors. It is therefore sensible for developers to first consider using game developing tools that support networking features natively. Although, looking for "the best" development tool is unreasonable, as this is a matter of personal preference. Before being able to investigate pros and cons between gamemaker and related game development tools, appropriate competitors must first be identified.

Note that GameMaker does natively support networking functionality for non-browser games. However we must remind ourselves of the aim of this project: creating a networking extension to improve GameMaker's functionality **for developing 2D browser games**. For this reason, this paper will consider GameMaker to be a development tool for browser games.

Unbiased sources were used in order to find the most related software for these purposes. These sources collect user feedback for the browser-game development tools. Their ratings were established by allowing users to criticise tools based on their personal experience, and apply a score. The review system of html5gameengine.com[1] represents that of the Google Play Store[18] and the Apple App Store[19], and may therefore be a somewhat reasonable method for finding the more commonly used tools.

developer.mozilla.org[2] (world rank 198 by alexa.com[3]) hosts a list of tools titled "HTML5 game engines", and holds suggestions by 26 developers each with different backgrounds in 2D browser game development. gamepix.com[4] hosts a list of similar tools, but is less detailed about their sources.

Table 3.1 was carefully constructed by combining the best rated 2D browser-supported game developing tools as advertised by the communities for the instances mentioned above.

[1] https://html5gameengine.com/
[2] https://developer.mozilla.org/en-US/docs/Games/Tools/Engines_and_tools
[3] http://www.alexa.com/
[4] http://www.gamepix.com/blog/the-big-list-of-2d-html5-games-engines/
[5] Advertised at developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Games/

Tool	Score	Voters	Mozilla[5]	Gamepix[6]	Native GUI
Pixi.js	5.0 / 5	50	yes	yes	no
Phaser	4.5 / 5	129	yes	yes	yes
EaselJS	4.5 / 5	63	no	yes	no
Crafty.js	4.5 / 5	18	yes	yes	no
Construct 2	4.0 / 5	136	yes	yes	yes
GameMaker	4.0 / 5	59	no	yes	yes
Quintus	4.5 / 5	29	no	yes	no
Unity	n/a	n/a	yes	yes	yes

Table 3.1: Some of the highest rated and most promoted 2D browser-game development tools

Tool	Topics	Community main page
Pixi.js	1455	http://www.html5gamedevs.com/forum/15-pixijs/
EaselJS	738	http://stackoverflow.com/questions/tagged/createjs[7]
Quintus	n/a	https://plus.google.com/communities/104292074755089084725
Crafty.js	1666	https://groups.google.com/forum/#!forum/craftyjs
Phaser	8793	http://www.html5gamedevs.com/forum/14-phaser/

Table 3.2: Displaying the community activeness for each of the mentioned tools, along with their corresponding location.

3.0.1 Pixi.js, EaselJS, Quintus, Crafty.js and Phaser

Despite Pixi.js, EaselJS, Crafty.js, Phaser and Quintus being legitimate tools for developing games, they are merely JavaScript frameworks and libraries to assist developers when creating raw JavaScript games. Therefore, if networking is expected to be part of a browser game, these tools will also require additional resources such as WebSocket for implementing this feature.

Their community is relatively small and scattered across the web. The forums for these tools are hosted on third-party websites and make it therefore difficult to measure their size and activity accurately (see table 3.2).

Although the five tools assist in the development, they do not compare with GameMaker as their user base is many times smaller. A GUI is also not provided, forcing developers to use a standard programming IDE.

3.0.2 Construct 2

Construct 2 is one of the main contenders. It provides its own GUI and shares similar ideals as those of gamemaker: simplifying the development for novice programmers.

Tools/Engines_and_tools

[6] Advertised at http://www.gamepix.com/blog/the-big-list-of-2d-html5-games-engines/

[7] Statement of community moving to stackoverflow: http://community.createjs.com/

This is done by supporting inbuilt D&D features. Code programming can also be used after installing a required extension. It is a generally well known tool with a community of over 204,000 registered users, responsible for more than 529,000 posts for 103,000 topics[8].

In April 2014, release r164 to r168 updated the engine with networking features by allowing users to create a said "network object"[9]. This object is pre-defined with "features to develop real-time online multiplayer games", provided using WebRTC DataChannels (UDP) for P2P connections.

It is a very well established update with optimisation support included, eg. by providing preliminary setups for "NAT traversal to connect through common router/network setups", "Interpolation and extrapolation modes to ensure smooth in-game motion", "Support for lag compensation" and more. Administrator at Scirra, Ashley G. states however that "even though Construct 2's Multiplayer object takes care of many of the complexities, ..., it is important to understand how multiplayer online games fundamentally work"[20].

3.0.3 Unity

In the world of browser-game development, Unity is by far the most acknowledged tool. It comes with a professional GUI, allowing developers to create games using D&D as well as with programming languages C# and JavaScript[10].

Networking features are supported as a D&D class[11], but can also be used by installing the "WebSocket-Sharp" plugin in C#[12].

Currently there are roughly 4,500,000 users registered to the Unity forum[13], however only a small percentage of this is active in 2D development as this only hosts 5,600 topics[14].

This is most likely due to the fact that Unity is aimed towards experienced 3D game-developers and offers much overhead when only using its 2D capabilities. In the poorly populated 2D section of the forum, replies such as "have you considered using GameMaker?"[15] are therefore a common response.

https://unity3d.com/public-relations

4,500,000 registered users

http://forum.unity3d.com/forums/2d.53/

[8] https://www.scirra.com/forum/
[9] https://www.scirra.com/manual/174/multiplayer
[10] http://docs.unity3d.com/ScriptReference/index.html
[11] http://docs.unity3d.com/ScriptReference/Network.html
[12] https://github.com/sta/websocket-sharp
[13] http://forum.unity3d.com/forums
[14] http://forum.unity3d.com/forums/2d.53/
[15] http://forum.unity3d.com/threads/where-to-start-2d-toolkit-or-platformer-pro.380770/

5,595 topics

licenses http://unity3d.com/unity/licenses cost https://store.unity3d.com/products/pricing

Chapter 4

Development

4.1 Design

4.1.1 Prior Considerations

Software Design Principles: The code needs to be coherent and follow software design principles where-ever applicable in order to support robustness, reliability, reusability, and understandability. This because it needs to be easily readable and understood by other developers that want to use the extension. It also needs to be maintainable for use by others and easily extendable whenever needed.

Client tasks: The client has to be executable in the browser. It should be an application or game which uses the extension in order to interact with other existing clients.

Extension tasks: The functions inside the extension should be easily callable from within the GameMaker interface. They should keep the "networking" knowledge requirement of the developer to a minimum. The extension also has to be able to execute in the browser as part of the client, and execute socket send- and receive commands while doing so.

Optimally, the networking functions should be similar to those of the native networking functions that would be supported on non-browser applications. This way developers merely need to translate the networking functions without having to re-think, re-structure or entirely re-write the functionality of their original code if they have already built an application that supports networking on different platforms.

Server tasks: The server must to mimic the architectural structure of the client. It has to keep track of the state of each of the connected clients. it should also act as

centralized controller that allows instances to interact with other instances. These instances are synchronized with those on the clients. Therefore, a change in the instance on the server should trigger a change for its representative instance in the clients.

Version Control: A vital part of this project that may not be ignored is to keep track of progress. During the development I have been subject to travelling and using a large variety of devices. The project is a combination of many smaller projects, and the progress of each of these needs to be carefully organised and controlled.

Therefore, ensuring consistency in these areas was taken care of by the choice of using a form of version control. A private Github[1] repository was therefore setup.

Socket.io versus WebRTC Transmission protocols are one of the contributors that will heavily characterise the network. Choosing the right system to handle the data transmissions is therefore crucial.

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.

The main focus in the extension is to match GameMaker's primary intention: making game development as fast and simple as possible. Focusing purely on this aspect leads to believe that using Socket.io is the best solution as it provides the most reliable form of communication. In contrast to WebRTC, developers will not have to worry about their packets being blocked at the firewall. They do will not need to worry about dealing with packet loss or having a shuffled order of packet arrival in their applications, and can instead purely focus on what it is they store into the packets that travels between the hosts.

4.1.2 Server and Client

4.1.2.1 Good Coding Practices

4.1.2.2 Interaction

4.2 Implementation

4.2.1 Server

The server is written in JavaScript, using Node.js in order to support a large number of concurrent client connections without overloading the memory (RAM).

[1] https://github.com/

As one of the main requirements of the project is to keep the development of networked browser applications as simple as possible, Socket.io is used in order to handle the send and receiving of messages. This way, the developer does not need to worry about complications that would otherwise arise when building the system with WebRTC such as difficulty during the implementation, packet loss and error handling, and UDP NAT traversal.

The core functionality of the server that takes care of incoming messages and connection requests is written in only a few lines and in simple terms functions as follows:

1. On an incoming connection request, a client instance is created in which initially the client's socket and client_id will be stored. The socket will be connected directly to the client machine that sent the connection request.
2. This instance is added to the list of already-tracked clients on the server
3. If at any time the server receives a message from this client, the server will read the purpose of this message (message_code e.g. ping, send-message, update-user)
4. The message_codes are directly mapped to an associated action
5. Depending on the action, the client instance will receive a function call that handles the message appropriately.

The client class is made to be synchronized with the state of its corresponding client, such that any change in the actual client application will also modify the representative client state in the server. This way, the developer no longer needs to worry about redundantly transmitting all variables to all other clients, including those variables that never changed since a previous transmission, because the server will still have these in memory. Another benefit to this is that the client is allowed to send smaller messages.

The server already has a template set-up, supporting features such as updating the client's state on the server, informing other clients with this client's state and broadcasting messages. These features can easily be modified and extended without with new functionality depending on any specific need the developer may have.

As the server keeps track of all the clients that are connected to it, clients can easily message specific clients simply by triggering a function call between any clients. Also the effect of the function is to inform the actual client with a message.

4.2.2 Client

4.2.3 The Extension

GameMaker is unable to use native support for establishing a connection to other hosts. This is where the extension comes in play. It is a set of JavaScript functions that can be called from inside the GameMaker GUI and provide the actual networking functionality. It consists of three components:

1. An MIT-licensed opensource socket.io JavaScript library provided by LearnBoost: socket.io.js (documentation [2])
2. A set of networking functions, located in GMWebNet.js
3. A mapping from GML functions to the networking functions

The socket.io library is used for providing the core networking functionality. This is required by the networking functions in order to do the transmission and receiving of packets.

GMWebNet.js contains a set of networking functions and is made to run in parallel with the GameMaker application. It is designed to take care of its own socket details, memory buffer and incoming messages. Developers do not need to modify this file. It merely provides the foundation on top of which any application-layered implementation can be built.

A simple connection function `connect(server_ip, server_port)` opens a socket connection with the server located at a given IP address and port number. This socket is immediately used in order to send a SYN request to the server, to which the server is expected to respond with a uniquely assigned client-id for the client. This unique client-id is received in the callback and will be caught in GameMaker as soon as it arrives. The function also returns this socket so that it can be used for communication with the server after initialization.

Its disconnection counterpart `disconnect(socket, value)` sends a message to the server, informing it that the socket is disconnecting. The server is expected to interpret this message as a user that is leaving. After sending the message, the socket is disconnected on the client-side and removed from the memory.

The function `send_message(socket)` sends whatever packet is stored in the buffer through the socket to the server. This packet can be pushed onto by using the `buffer_write(value)` function.

When a message arrives, it gets automatically stored in the messages array. `receive_message(socket)` stores this message into the buffer. And the buffer content can then be read from by the application when calling `buffer_read()`.

When at any time, the buffer needs to be reused for sending a new message, `buffer_clear()` must first be called to clear the content in the buffer.

mapping Creating a mapping can be done directly in the GUI of GameMaker as displayed in figure 4.1. After creating a new extension setup in the extensions directory, the previously specified extension files can be referenced. In each of the files, functions can be added that map GML functions to extension functions. Figure 4.2 is part of the mapping xml file that displays how it looks in detail.

[2] https://docs.omniref.com/js/npm/socket.io/0.8.4

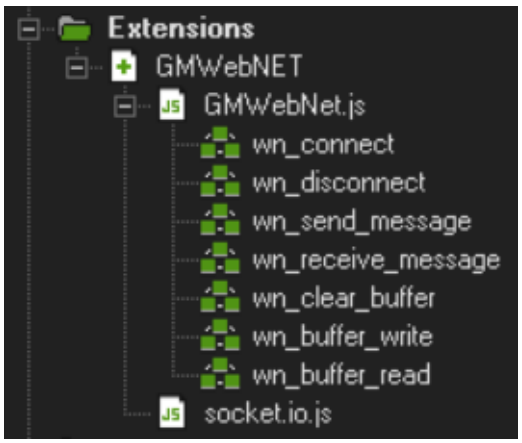

Figure 4.1: View inside the GUI that visualises the extension function mapping setup.

```
<function>
  <name>wn_connect</name>
  <externalName>connect</externalName>
  <kind>11</kind>
  <help>
    wn_connect(server_ip, server_port)
    -- Connect to a server at ip (address) and port. Returns socket-ID
  </help>
  <returnType>2</returnType>
  <argCount>2</argCount>
  <args>
    <arg>1</arg>
    <arg>1</arg>
  </args>
</function>
```

Figure 4.2: External function `connect` from JavaScript extension file located in GMWebNet.js, mapped to a GML function `wn_connect` located in the client application. The function takes 2 arguments, each of type String (id = 1), and returns a Double type (id = 2)

The natively supported networking functions can easily be translated into the the browser-compatible functions defined by the extension as displayed in table 4.1. The following purposes are included:

- initiating a connection
- creating a packet in the buffer
- sending a packet
- spot incoming messages

- read incoming messages
- disconnect hosts

4.3 Applications Developed with the Extension

4.3.1 Benchmark Application

4.3.2 Real Game Application

4.3.3 Developer Template

Function Translations

Creating a socket and connecting to a server

Native

```
socket = network_create_socket(network_socket_tcp);
network_connect(socket, ip_address, port);
```

Extension

```
socket = wn_connect(ip_address, port);
```

Creating a new packet in the buffer and sending it

Native

```
buffer = buffer_create(size, type, byte_alignment);
buffer_write(buffer, buffer_size, value);
network_send_packet(socket, buffer, buffer_get_size(buffer));
```

Extension

```
wn_buffer_clear()
wn_buffer_write(value)
wn_send_message(socket)
```

Spot an incoming packet and start reading it

Native

```
new_messages = socket_read_message(socket, buffer)
if (new_message) {message = buffer_read_uint8(buffer);}
```

Extension

```
new_messages = wn_receive_message(socket)
if (new_message) {message = wn_buffer_read();}
```

Disconnecting the hosts

Native

```
network_destroy(socket);
```

Extension

```
wn_disconnect(socket, disconnect_message_id)
```

Table 4.1: Function Translations from natively supported functions to browser-supported functions. Notice that the extension itself takes care of implementation details for the buffer, buffer size and the connection type (TCP).

Chapter 5

Network Extension Evaluation

5.1 Controlled Network

Controlled experiments were conducted in order allow prediction of server behaviour when loaded under similar pressure in future occasions.

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 bandwidth [1]. 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.

5.1.1 Concurrent Connections

It is important to know how many clients a server can host simultaneously. The following experiments evaluate the server performance in terms of server CPU usage, CPU time and RSS with regard to the number of concurrently connected clients.

Setup

- 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 resulting values are averaged over a 2-minute run duration.

[1] http://www.ispreview.co.uk/index.php/2015/02/ofcom-average-uk-home-broadband-speeds-slowly-reach-23mbps.html

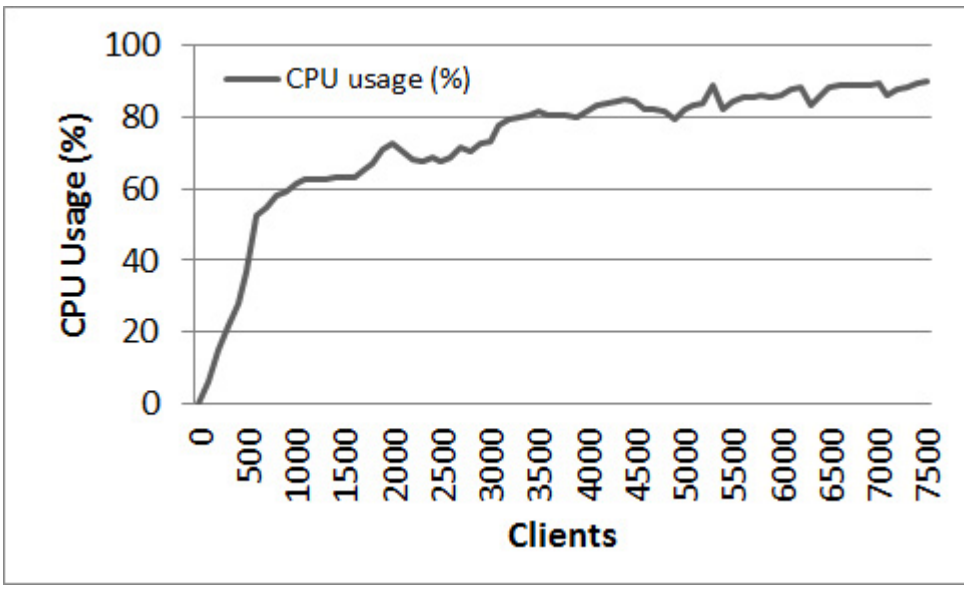

Figure 5.1: Displaying the CPU usage for the server process in percent.

- After each run, the number of clients simultaneously connected to the server increases by 100.

Figure 5.1 indicates a logarithmic trend in the CPU usage on the server. When the CPU usage reaches over 80 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.

Due to The server stores connection details for each client directly in the RAM.

5.1.2 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.

Environment Setup

- 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.

Dependent variables

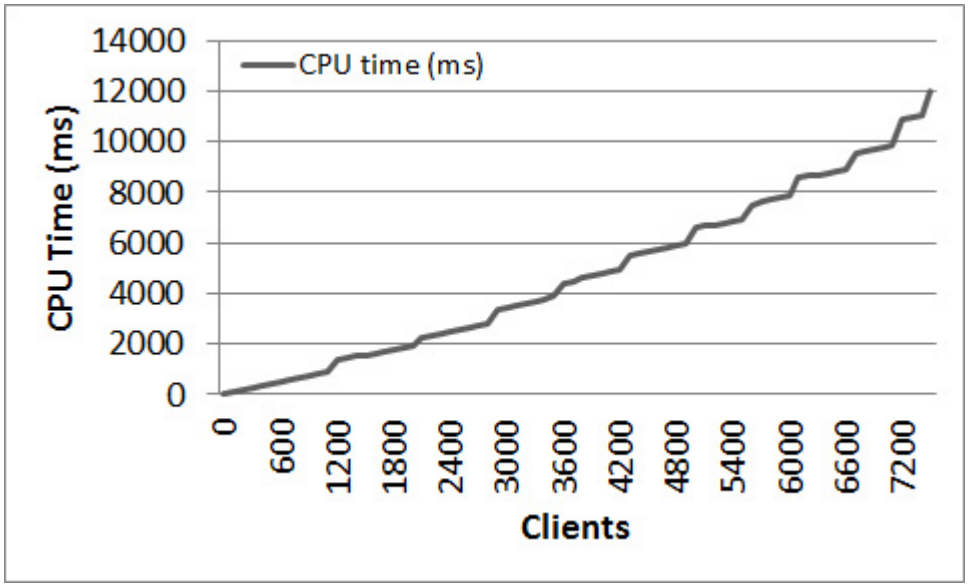

Figure 5.2: CPU time in milliseconds, displaying the amount of time required for the server to process the clients.

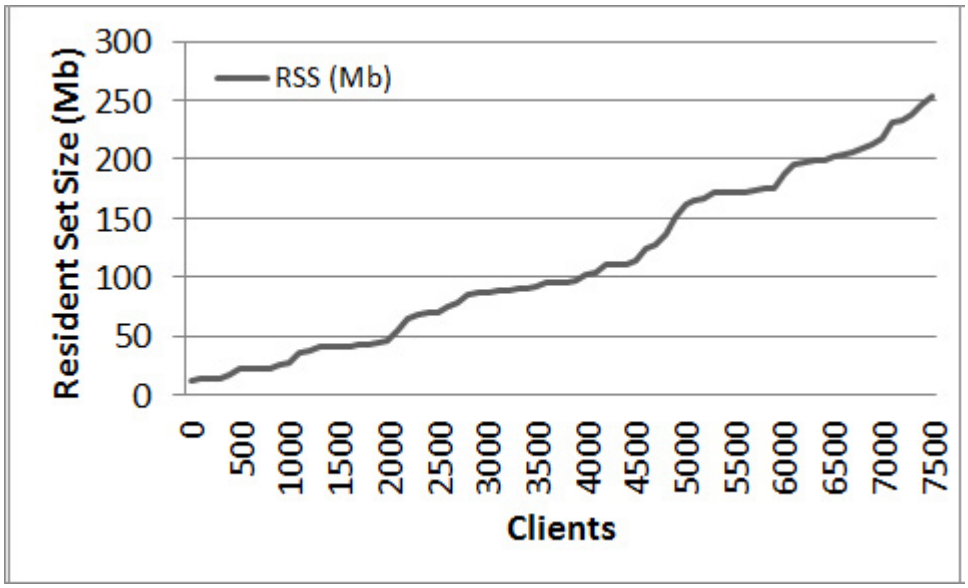

Figure 5.3: Resident set size in Megabit, showing the portion of RAM that is occupied by the server process.

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

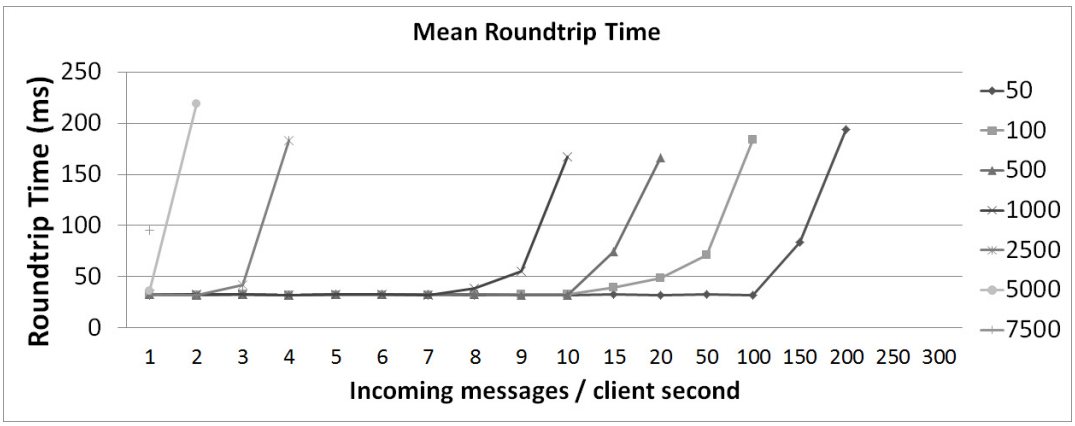

Figure 5.4: The mean roundtrip time of all the messages that pass through the server. As expected, with few concurrent clients connected to the server, the server manages to broadcast many more messages.

5.2 Real Network Results

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

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

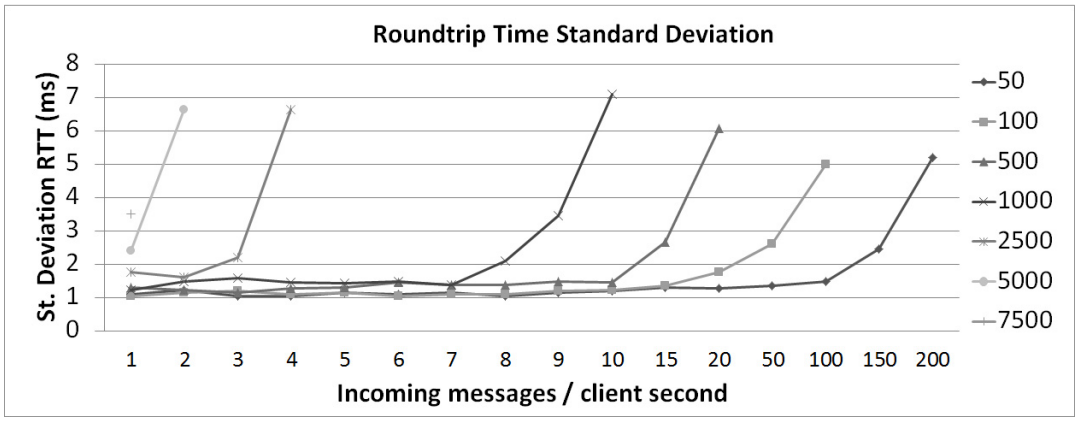

Figure 5.5: Standard deviation of the message roundtrip times.

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.

5.2.1 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.

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.
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.

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
Std RTT	0.4ms	2.9ms	6.8ms	3.36ms	43.6ms

Chapter 6

Conclusion and Future Work

6.1 Conclusion

6.1.1 Comparison of the Extended GameMaker Functionality with Related Work

6.1.2 Criticism on the Implementation and Design Decisions

6.2 Note to developers

6.3 Future Improvements

[21] [2] [22] [10] [23] [16] [9] [12] [13] [14] [5] [6] [4] [11] [1] [24] [25] [26] [7] [1] [8] [15] [17] [27] [28] [18] [19] [29] [30] [31] [20] [3] [32] [33]

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] Jeremy Brun, Farzad Safaei, and Paul Boustead. *Fairness and playability in online multiplayer games*. PhD thesis, University of Wollongong, 2006.

[3] David Walsh. Websocket and socket.io. https://davidwalsh.name/websocket, 29 November 2010. [Accessed: 2016-02-24].

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

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

[6] 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].

[7] Ilya Grigorik. *High Performance Browser Networking*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2013.

[8] Martin Kirkholt Melhus. *P2P Video Streaming with HTML5 and WebRTC*. PhD thesis, Norwegian University of Science and Technology, Trondheim, June 2015.

[9] 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].

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

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

[12] 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].

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

[14] 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].

[15] Thomas Claburn. Gamemaker promises drag-n-drop game creation. *Informationweek*, May 2012.

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

[17] Llopis Noel. Gamemaker studio v1.1.7. *Game Developer*, 20(1):40, January 2013.

[18] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. Why people hate your app: making sense of user feedback in a mobile app store. *KDD '13: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 13:1276–1284, 2013.

[19] Jin Hyung Kim, Inchan Kim, and Ho Geun Lee. The success factors for app store-like platform businesses from the perspective of third-party developers: An empirical study based on a dual model framework. *PACIS Conference Proceedings*, 2010.

[20] Ashley Gullen. Multiplayer tutorial. https://www.scirra.com/tutorials/892/multiplayer-tutorial-1-concepts/page-1, March 2014. Scirra, creators of Construct 2.

[21] Alan Edwardes. Multiplayer networking in a modern game engine. Project Writeup, 2014.

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

[23] 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.

[24] 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.

[25] 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*, 3(1), 2013.

[26] 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.

[27] Alessandro Alinone. Optimizing multiplayer 3d game synchronization over the web. *Lightstreamer*, October 2013.

[28] Eric Li. Optimizing websockets bandwidth. *Multiplayer*, September 2012.

[29] Jason Lee Elliott. *HTML5 Game Development with GameMaker*. Packt, 35 Livery Street, Birmingham B3 2PB, UK, 2013.

[30] Barry W. Boehm. Seven basic principles of software engineering. *Journal of Systems and Software*, 3(1):3–24, March 1983.

[31] Klaus Pohl, Gunter Bockle, and Frank van der Frank. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, Springer-Verlag Berlin Heidelberg, 2005.

[32] Yevhen Shylo. Network address translation. *Scientific Report - Technological Conference*, 3(1):79, November 2015. http://www.dut.edu.ua/uploads/n_2205_50283608.pdf.

[33] Phil Edholm. Where are the real webrtc apps? http://www.webrtcworld.com/topics/from-the-experts/articles/375465-where-the-real-webrtc-apps.htm, 2014. [Accessed: 2016-01-17].