A Network Extension for GameMaker HTML5

Teun Kokke
s1242775

Undergraduate Dissertation
Software Engineering
School of Informatics
University of Edinburgh

2016

Abstract

summarising the report

TODO

- 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

- 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

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

- D&D - Drag and Drop

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 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].

2.2 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" [3], it keeps the connection open [4]. 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.

2.3 UDP

UDP is a connectionless transmission protocol that, unlike TCP, provides a "minimal, unreliable, best-effort message-passing transport to applications" [5]. It provides no

guarantees for delivery and no protection from duplication. Despite this clear down-side, 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.

2.4   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 [6].

2.5   WebSocket

WebSocket is a protocol that "provides a method to push messages from client to server efficiently and with a simple syntax" [7]. 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" [8]. 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.

2.6   Socket.io

Socket.io is an event-driven JavaScript library that can be used both on the client's browser, and a server [9]. It supports features that allow sending and receiving data using the WebSocket protocol, without interruption of the code flow [10, 11]. For this reason, it is **often used in combination with Node.js**.

2.7   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" [12]. Although certain web applications

have already been created with WebRTC (mainly for P2P video streaming [13]), 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 [13], makes UDP many times **more complicated** for developers to handle efficiently.

2.8 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 [1]. 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 [14].

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" [15]. **The element can thus be used to draw graphics in JavaScript with the "Canvas API"** [16].

2.9 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 [17].

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" [17]. 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 the (Node.js) event loop. **This therefore allows servers running Node.js to maintain thousands of concurrent connections without running out of RAM memory** [18, 19], as opposed to the traditional, less scalable servers.

---

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

2.10   GameMaker

GameMaker by YoYoGames is a software creation tool, primarily with the aim to simplify, teach, and speed up game and application development for novice software developers and programmers. There have been several hits on the market for games developed with Gamemaker such as "Reflections", "Rick O'Shea" and "Simply Solitaire" [20].

Developing applications and games in GameMaker is cheap, simple to learn and flexible to use, making the software demanded by small teams, novice developers and even professionals [21]. 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 [20].

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 [22].

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 [23]. 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 [24]. It is a basic setup that merely broadcasts messages as they come in.

The main GameMaker community forum is hosted directly by YoYoGames [25], 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 [26] which hosts an additional 557,000 posts for 56,000 topics for almost 18,000 members.

2.11   Quality Attributes

Quality attributes are factors that represent non-functional requirements of a software system. This entails performance details on how the software behaves at run-time and terms of how well is designed. Definitions to some quality attributes that are especially important to this project are listed in table 2.1.

performance:	An indication of the responsiveness of a system to execute any action within a given time interval.
managability:	How easy it is to manage the application.
reusability:	The capability for components and sybsystems to be suitable for use in other applications and scenarios.
maintainability:	The ability to undergo changes such as changing features and functionality with a degree of ease.
Conceptual Integrity:	The consistency and coherence of the overall design, including the way that components or modules are designed, as well as coding style and variable naming.
Reliability:	The probability that a system will not fail to perform its intended functions.
Security:	The capability of a system to prevent malicious or accidental actions outside of the designed usage.
Scalability:	The ability to handle increases in load without impact on the performance on the system.

Table 2.1: Quality attributes with their definition as described by Microsoft [27].

2.12   Fairnesss and Playability in Online Multiplayer Games

more detail on Network Fairness, causes and fixes to unfairness

2.13   Multiplayer Networking in Modern Game Engines

game state maintained through a limited set of parameters

Chapter 3

Related Tools

3.1 Identifying competitors to GameMaker

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 [28] represents that of the Google Play Store [29] and the Apple App Store [30], and may therefore be a somewhat reasonable method for finding the more commonly used tools.

developer.mozilla.org [31] (world rank 198 by alexa.com [32]) 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 [33] 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.

Tool	Score	Voters	Mozilla[1]	Gamepix[2]	Native GUI
GameMaker	4.0 / 5	59	no	yes	yes
Construct 2	4.0 / 5	136	yes	yes	yes
Unity	n/a	n/a	yes	yes	yes
Pixi.js	5.0 / 5	50	yes	yes	no
EaselJS	4.5 / 5	63	no	yes	no
Quintus	4.5 / 5	29	no	yes	no
Crafty.js	4.5 / 5	18	yes	yes	no
Phaser	4.5 / 5	129	yes	yes	yes

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

3.2 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. 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 in 103,000 topics [34].

In April 2014, release r164 to r168 updated the engine with networking features by allowing users to create a said "network object" [35]. 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"[36].

3.3 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 [37].

---

[1]Advertised at developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Games/Tools/Engines_and_tools

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

Networking features are supported as a D&D class [38], but can also be used by installing the "WebSocket-Sharp" plugin in C# [39].

Currently there are roughly 4,500,000 users registered [40] to the Unity forum [41], however only a small percentage of this is active in 2D development as this only hosts 5,600 topics [42].

This is most likely due to the fact that Unity is aimed towards more experienced 3D game-developers and provides 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?" [43] are therefore a common response.

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

3.4   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).

Tool	Topics	Community main page
Pixi.js	1455	http://www.html5gamedevs.com/forum/15-pixijs/
EaselJS	738	http://stackoverflow.com/questions/tagged/createjs[3]
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.

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.

Chapter 4

Development

4.1 Prior Considerations and Design Decisions

Version Control: A vital part of this project 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 making the choice of using a version control system. A private Github [44] repository was therefore setup.

4.1.1 Non-Functional Requirements and Desirable Quality Attributes

The project consists of a client, networking extension and server. Each of these systems has to conform to and prioritize certain non-functional requirements (see table 2.1).

The client: The client performance is only of secondary value in the project as the aim with the extension is that other developers can develop their own client applications. The only thing that is expected to be the same with the client written in this project is the way the networking functions are called. However, managability, reusability and conceptual integrity are all very important for the developer to be able to understand, modify and extend the client, transforming it into their own version.

The extension: For the extension, performance becomes crucial as it connects the client and the server. It is therefore important to measure how well the extension performs in terms of the networking capabilities it needs to provide. It is not expected to be read or modified by any developer, as it already supports all features required to make a live networked application to run. Any modification involving further optimization for the network would potentially complicate the system to the point where a different game development tool should be considered. Reliability is also an important

attribute, as when the extension fails to transmit or receive messages, the developer would have to learn about the underlying networking systems. This would contradict the main goal of the project: keeping the GameMaker simple to learn and flexible to use, even for novice developers.

The server: The server matches the desired qualities of both the client and the extension, in the sense that it needs to be easily understood by the developer. The developer should be able to manage and modify the server settings, maintain the functionality and reuse existing methods without having to fear breaking the core structure. It needs to be scalable as the performance might decrease when large numbers of clients may wish to be connected.

Also security becomes an important aspect of the server as it may receive unintended or malicious requests, it is however not considered to be of primary importance in this project and developers may prefer to implement their own sercurity features at the application level. Such feature could involve including secret keys in the interaction messages between the client and server in order to ensure authenticity of these messages and their senders.

4.1.2   Functional Requirements and Tasks

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. The interaction consists of transmitting and receiving small messages containing variables, due to which the network bandwidth is not counted as an important factor.

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 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 should in turn be synchronized with their referenced instances on the clients. Therefore, a change in the instance on the server should trigger a change for its representative instance in the clients.

4.1.3 Further considerations

Server Technology   Using the right server technology is of crucial importance. Games and browser applications are often subject to large volumes of concurrently connected clients. In this scenario, scalability is therefore one of the most important non-functional requirements.

Node.js is especially well known to do exactly that: supporting a large number of concurrent client connections without overloading the memory (RAM). Using JavaScript combined with Node.js is therefore considered to be the best option for the server-side to be written in.

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 primarily 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 UDP NAT traversal, packet loss, error handling or having a shuffled order of packet arrival in their applications, and can instead focus directly on what it is they store into the packets that travels between the hosts.

4.2 Implementation

4.2.1 Server

The core functionality of the server that takes care of incoming messages and connection requests is written in only a few components. The general interaction flow is displayed in figure 4.1. Each component is aimed to be easily maintained by assigning it with only a single responsibility. These responsibilities are clarified directly in the naming of the variables and methods in order to support conceptual integrity of the code.
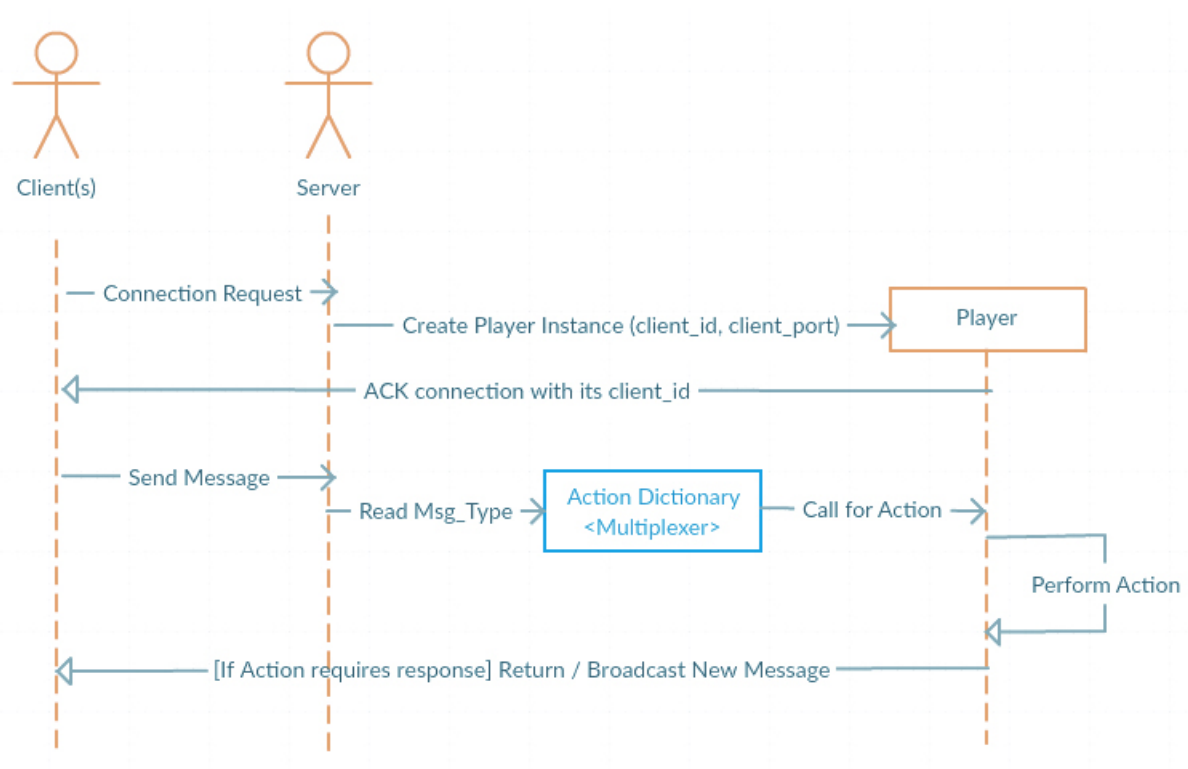
Figure 4.1: Sequence diagram displaying the general server logic. Note that clients can asynchronously and repeatedly send messages to the server after having received their server acknowledgement (ACK) packet. The `Msg_Type` can be anything such as `chat_message`, `update_user` and `client_disconnected`, and the action taken by the server is specified by the player object method, referenced by the action dictionary on the server.

In code, the server logic behaves 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. (figure 4.2)

2. This instance is added to the list of already-tracked clients on the server (figure 4.2)

3. If at any time the server receives a message from this client [1], the server will read the purpose of this message (message_code e.g. ping, send-message, update-user) (figure 4.2)

4. The message_codes are directly mapped to an associated action (figure 4.2)

5. Depending on the action (figure 4.3 defined in 4.6), the client instance will receive a function call that handles the message appropriately. (figure 4.4)

---

[1]Message types are defined in fig. 4.6

```
/* CORE FUNCTIONALITY */
// on new incoming connection-request..
io.sockets.on('connection', function(client_socket) {

    //Create New Player Instance (and append to players list)
    var client_id = controller.clients.length;
    var obj_player = new Player(client_id, client_socket);
    controller.clients.push(obj_player);


    // handle incoming messages from clients
    client_socket.on('message', function(data) {
        buffer          = data;          //store data into the buffer
        message_code    = bufferread(); //read the message type
        action[message_code](obj_player);
    });
});
```

Figure 4.2: Core functionality of the server network setup, responsible for connecting new clients, creating reference instances with their details connection details (1, 2), listening to their incoming messages on the server (3) and handling them accordingly (4, messages codes are defined in messages.js. see figure 4.6).

```
/* Network Logic Handling */
action = {};
var msgs = MESSAGES['C2S'];
action[msgs.initial_client_details] = function(player){
    username = bufferread();
    player.set_username(username);
};
action[msgs.client_disconnects] = function(player){
    console.log("logout");
    controller.clients.splice(player, 1);
    player.disconnect();
};
action[msgs.chat_message] = function(player){
    var chat_message = bufferread();
    player.send_message(chat_message);
};
action[msgs.ping] = function(player){
    var ping_sendtime   = bufferread();
    player.return_ping(ping_sendtime);
};
action[msgs.client_count] = function(player){
    player.return_clientcount();
};
action[msgs.update_user] = function(player){
    player.update();
};
```

Figure 4.3: A dictionary storing actions for a player instance to perform.

```
Player.prototype.disconnect = function() {
    msg_type = MESSAGES['S2C'].client_disconnected;
    clearbuffer();
    bufferwrite(msg_type);
    bufferwrite(this.client_id);
    broadcast();
};
Player.prototype.send_message = function(message) {
    msg_type = MESSAGES['S2C'].chat_message;
    clearbuffer();
    bufferwrite(msg_type);
    bufferwrite(this.client_username);
    bufferwrite(message);
    broadcast();
};
Player.prototype.return_ping = function(send_time) {
    msg_type = MESSAGES['S2C'].ping;
    clearbuffer();
    bufferwrite(msg_type);
    bufferwrite(this.client_id);
    bufferwrite(send_time);
    sendmessage(this.socket_id);
};
```

Figure 4.4: Low level methods the player instance on the server can perform. These methods are called by the incoming message block (figure 4.2 point 4), defined in the action dictionary (figure 4.3)

The client class (Player) 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 (figure 4.5). 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 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.

```
function Player(client_id, socket_id) {
    /* sets client_id and socket_id for the client, and
       returns the assigned client_id back to the client */
    self = this;
    this.client_id = client_id;
    this.socket_id = socket_id;

    // This object represents a player in the game.
    // Make sure the variables match those of the client player object that you want to
    this.client_username = null;    //user name
    this.client_spriteID = null;    //user appearance
    this.x = null;                  //x coordinate of user character
    this.y = null;                  //y coordinate of user character
    this.hspeed = null;             //horizontal speed of user
    this.vspeed = null;             //vertical speed of user
    this.mouse_x = null;            //user mouse x-coordinate
    this.mouse_y = null;            //user mouse y-coordinate
    this.td_direction = null;       //direction to which user is looking
    this.room_name = null;          //room the user is in
    this.armor_top = null;          //equipment user is wearing
    this.level = null;              //level of the user

    _initial_reply();

    function _initial_reply() {
        clearbuffer();
        bufferwrite(MESSAGES['S2C'].confirm_client_id);
        bufferwrite(self.client_id);
        sendmessage(self.socket_id);
    }
}
```

Player instance values from client (client_id, socket_id)

forward the assigned client_id and socket_id to this client for the newly connected user

Figure 4.5: Player class matching variables of the GameMaker client player object class. Upon instantiation, the constructor automatically returns a message to the referenced client with it's unique identity defined by the server.

```
var MESSAGES = {
    "S2C": {
        "confirm_client_id": 1,
        "send_client_name": 2,
        "client_disconnected": 3,
        "chat_message": 4,
        "ping": 5,
        "client_count": 6,
        "request_count": 7,
        "update_user": 8
    },
    "C2S":{
        "initial_client_details": 1,
        "client_disconnects": 2,
        "chat_message": 3,
        "ping": 4,
        "client_count": 5,
        "request_count": 6,
        "new_virtual_client": 7,
        "remove_virtual_client": 8,
        "update_user": 9
    }
};
```

Figure 4.6: Server specified message tags. C2S refers to "Client-to-Server" and S2C refers to "Server-to-Client" messages.

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 Learn-Boost: socket.io.js (documentation [45])

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(socke`
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 4.7. 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 4.8 is part of the
mapping xml file that displays how it looks in detail.
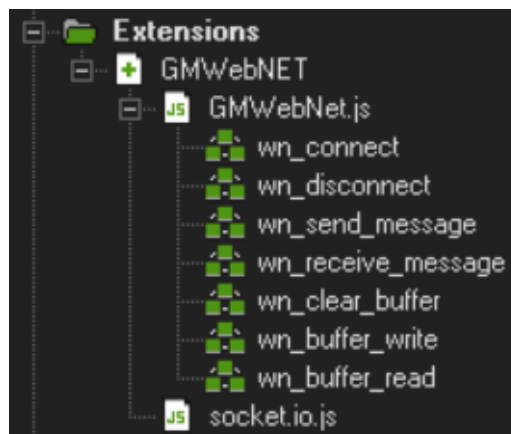
Figure 4.7: View inside the GUI that visualises the extension function mapping setup.

```
<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 4.8: 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

The benchmark application was the first application created using the networking extension. It initially consisted of a single instance repeatedly sending a ping request to the server at a fixed interval, out of which the RTT could be calculated.

```
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).

Since then, it has evolved into supporting a variety of instances (virtual clients), each being connected to the server through their own private connection. This way, it would allow for evaluation with many clients without being limited to a small amount of physical devices. The virtual clients are set to send messages at fixed intervals (ticks).

As the benchmarking application supports more than one virtual client that connects to the server, the benchmarker keeps track of the values of each of its virtual clients, calculating the averages, displaying these in a graph and creating logs for later evaluation.

On the top-left of the figure 4.9 the global data can be seen. This data only regards the virtual clients (not all clients that may be running on other devices).

The `CPU` value indicates the speed at which the CPU processes the code at runtime. This had to remain at a steady 60 in order to ensure no external variabilities would interfere with the experiments. Similarly, the `FPS` had to remain at a steady 30fps. This was the frame rate at which the application is drawing the GUI. If at any point the CPU or FPS would drop, this would indicate an external factor is affecting the program execution and will potentially corrupt the test results. Therefore, values that were collected when these values dropped would automatically be discarded.

The `clients` variable indicates the number of virtual clients the application has created.

`real_clients` indicates the number of clients that have successfully established a private connection to the server.

`global_clients` shows the user how many physical and virtual clients are connected to the server globally. This includes clients that are connected through other devices. This number is only updated on request by the user, as it depends on a request to the server and could reduce the accuracy of the experiments. Upon receiving this request, the server has to reply with the number of clients it is serving at that given time.

`average` and `deviation` refer to the average RTT and RTT deviation of the most recently sent 8-byte messages by each of the virtual clients.

`ping-rate/s` is the amount of 8-byte messages that are sent by each virtual client every second.

The `total req/sec` therefore indicates how many messages are sent out to the server every second by all of the virtual clients collectively.

`ping count` describes the size of the message history. 40 means that the graph will display the last 40 ticks worth of average RTTs of all the virtual clients.

This application was used in the evaluation experiments in order to test the system's capabilities, such as the applied stress on the server with different varieties of concurrent connections, certain amounts of messages that need to be processed simultaneously, as well as recording the effect on the RTT when the client is located in different areas in the world.
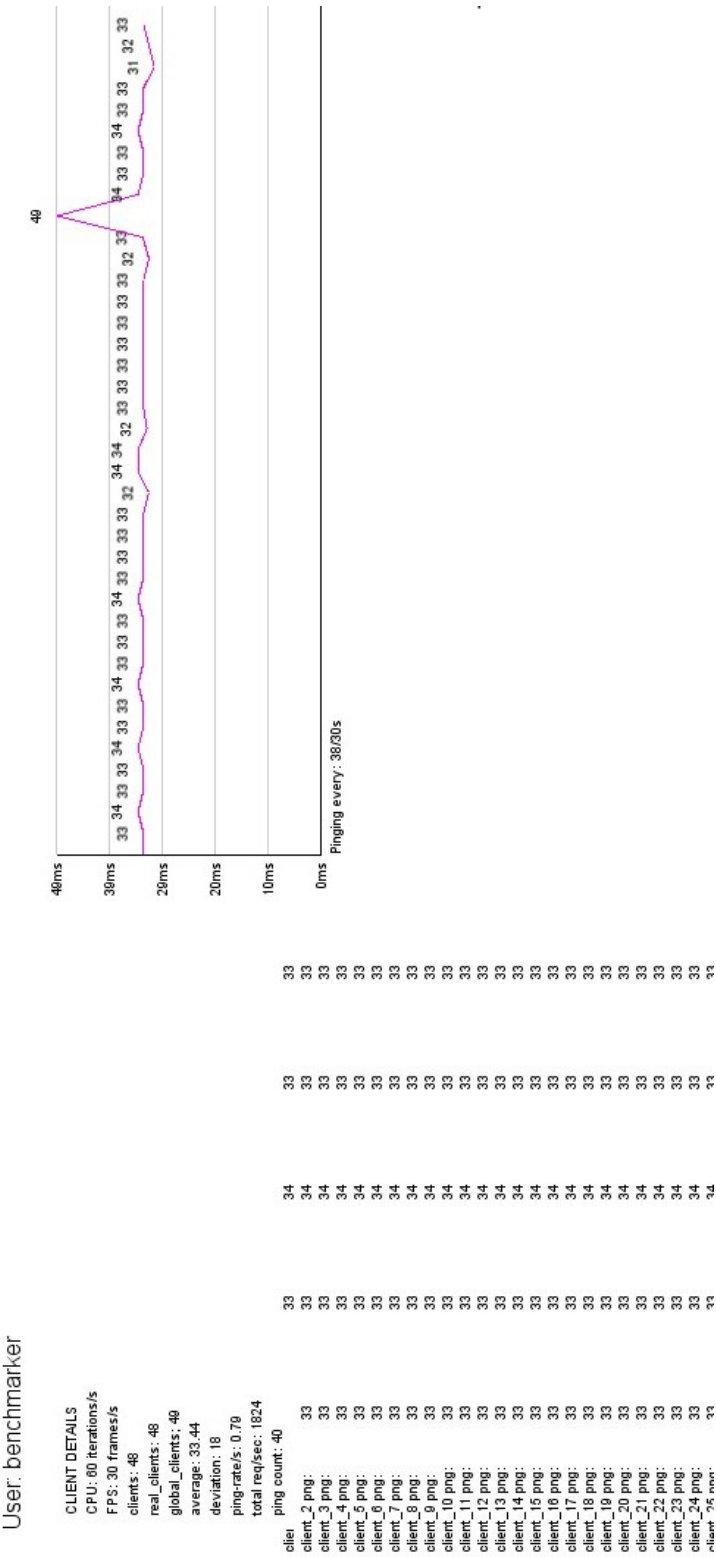
Figure 4.9: Displaying the GUI of the benchmarking application. On the left top the global data is available for the client. The table below displays the virtual clients and their 5 most recent roundtrip values. The graph on the top-right displays the average RTT over time for these clients. The colors of the image are inverted from the actual interface.

4.3.2   Real Game Application

4.3.3   Developer Template

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 entailed 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 [46]. 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, 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.

- 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. This seems strange. Although this effect may be explained when combined with the CPU time in figure 5.2.

The system aims to run the process at an optimal amount for CPU usage. As soon as it reaches this point, it will attempt to balance the usage by allowing more processing time for the process.
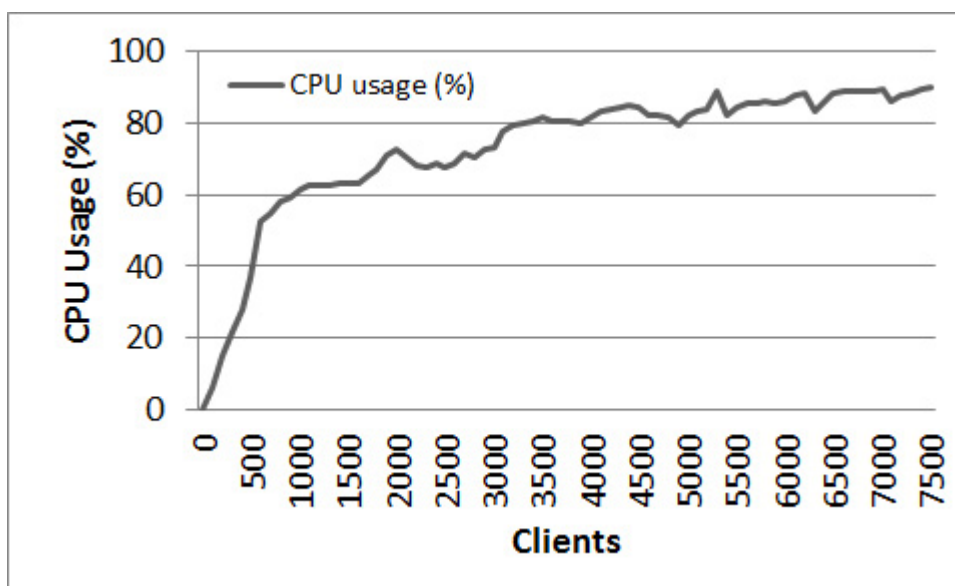

Figure 5.1: Displaying the CPU usage for the server process in percent.

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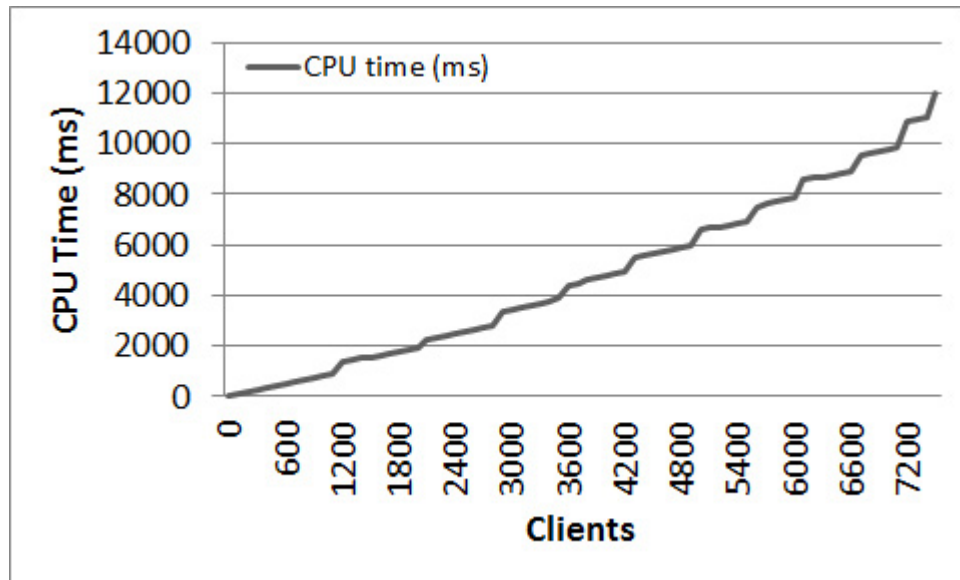
Figure 5.2: CPU time in milliseconds, displaying the amount of time required for the server to process the clients.
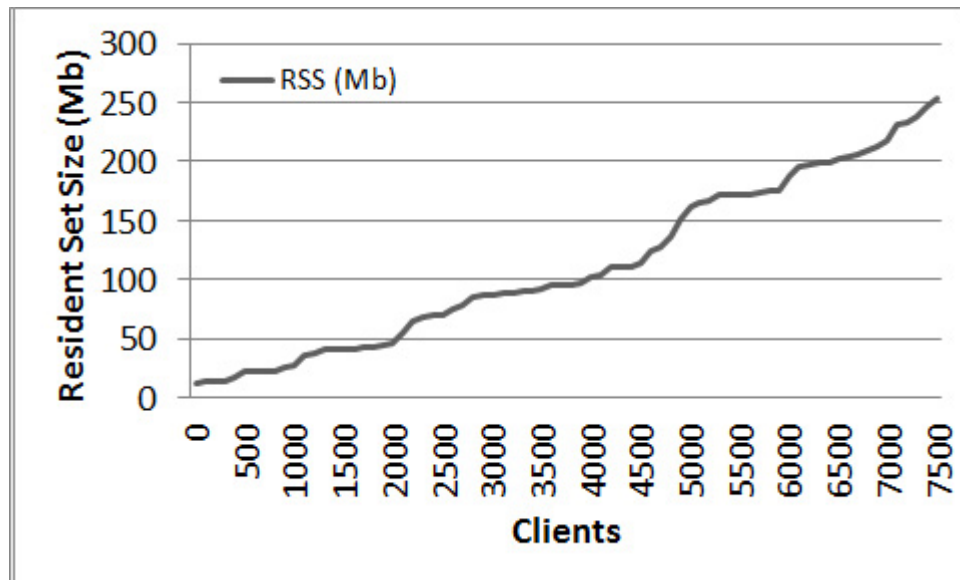

Figure 5.3: Resident set size in Megabit, showing the portion of RAM that is occupied by the server process.

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

- Mean roundtrip time between all clients

- Variance of the roundtrip time between all clients

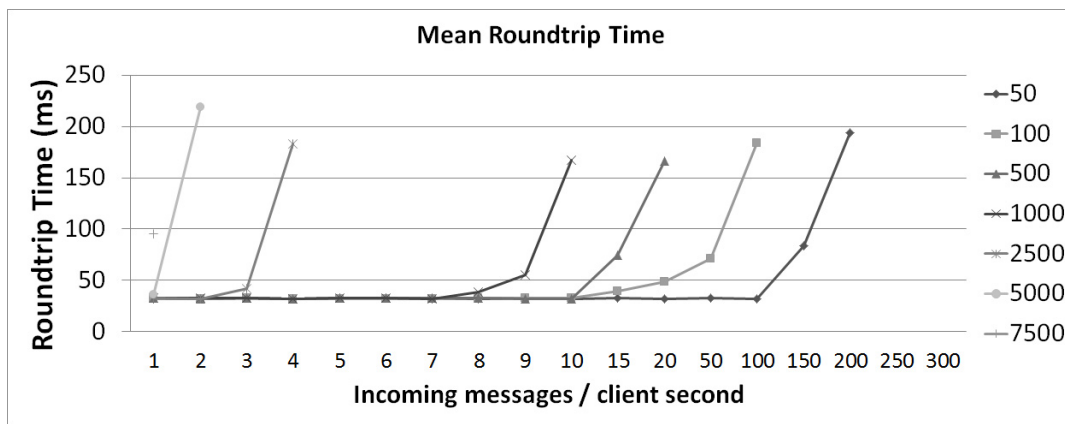- Standard Deviation of the roundtrip time between all clients

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.
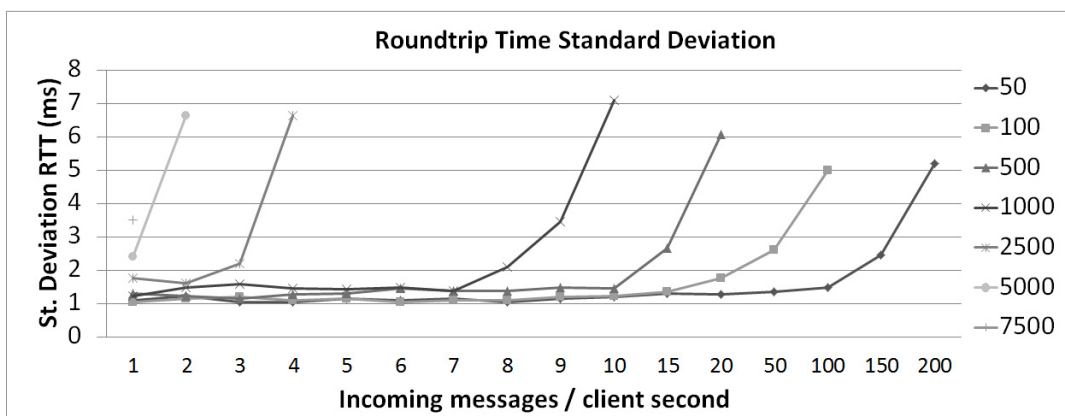

Figure 5.5: Standard deviation of the message roundtrip times.

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 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. During the experiments, logs were saved at the clients. After each experiment, these logs were forwarded for post-processing purposes. The roundtrip times represent the delay times between the packet-send event, and the packet-receive event by the same client for the same packet.

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: Six clients physically located in the United Kingdom, Hungary, France, Germany, Sweden and the Netherlands.

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

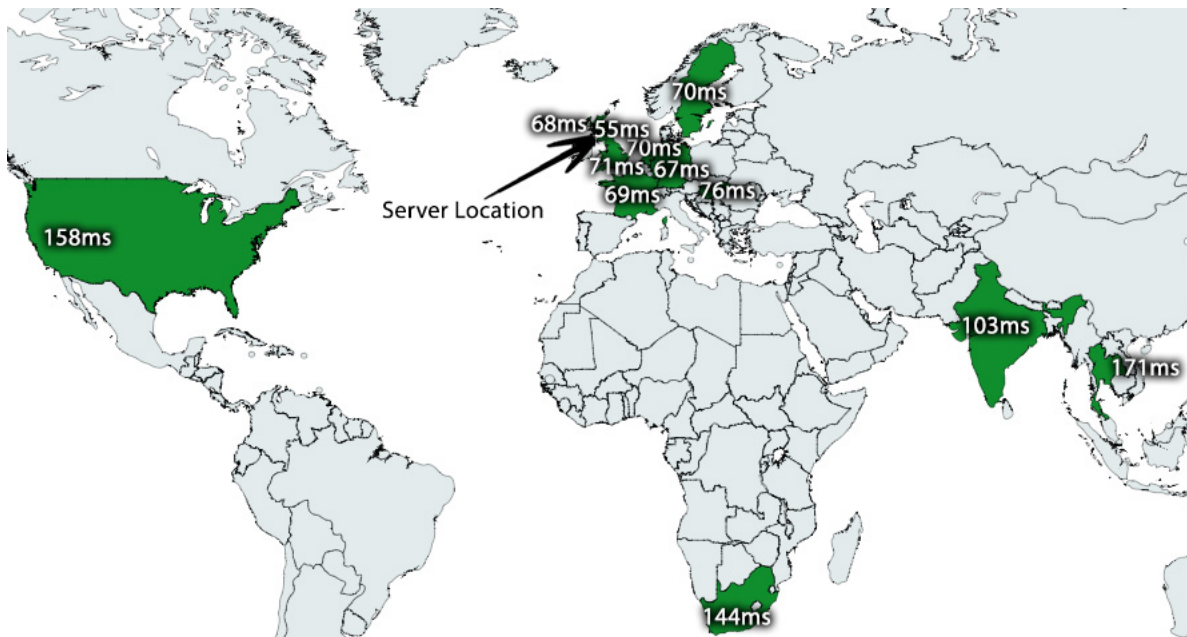**Results:** Average roundtrip times per location:

Figure 5.6: Worldmap displaying the tested locations and their according average recorded roundtrip times

Location	Average RTT
LAN	32ms
Edinburgh	55ms
Dundee	65ms
Germany	67ms
Glasgow	68ms
France	69ms
the Netherlands	70ms
Sweden	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

proper citations
[47] [2] [48] [15] [49] [21] [14] [17] [18] [19] [10] [11] [9] [16] [1] [50] [51] [52] [12] [1] [13] [20] [22] [53] [54] [29] [30] [55] [56] [57] [36] [7] [58] [59]

web url referrals
[3] [4] [5] [6] [8] [23] [24] [25] [26] [28] [31] [32] [33] [34] [35] [37] [38] [39] [41] [42] [43] [44] [45] [46] [40] [27]

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] Tcp 3-way handshake. http://www.inetdaemon.com/tutorials/internet/tcp/3-way_handshake.shtml. Accessed: 2016-03-02.

[4] Tcp open connection. https://tools.ietf.org/html/rfc793. Accessed: 2016-03-02.

[5] Udp connectionless. http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html. Accessed: 2016-03-02.

[6] Udp holepunching. https://tools.ietf.org/html/rfc5128#page-11. Accessed: 2016-03-02.

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

[8] Websocket two-way communication. https://tools.ietf.org/html/rfc6455. Accessed: 2016-03-02.

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

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

[11] 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].

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

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

[14] 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].

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

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

[17] 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].

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

[19] 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].

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

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

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

[23] Missing networking feature for browser export. http://docs.yoyogames.com/source/dadiospice/002_reference/networking/index.html. Accessed: 2016-03-02.

[24] Browser networking attempt for gamemaker html5. https://github.com/amorri40/39js. Accessed: 2016-03-02.

[25] Yoyogames community, forum. http://gmc.yoyogames.com/. Accessed: 2016-03-02.

[26] Dutch gamemaker community, forum. http://www.game-maker.nl/forums/. Accessed: 2016-03-02.

[27] Quality attributes. https://msdn.microsoft.com/en-us/library/ee658094.aspx#Reusability. Accessed: 2016-03-02.

[28] Review system for html5 game development tools. https://html5gameengine.com/. Accessed: 2016-03-02.

[29] 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.

[30] 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.

[31] Suggested html5 game engines through developer.mozilla.org. `https://developer.mozilla.org/en-US/docs/Games/Tools/Engines_and_tools`. Accessed: 2016-03-02.

[32] Alexa: Website rank indexing service. `http://www.alexa.com/`. Accessed: 2016-03-02.

[33] Gamepix: Html5 game engine developer suggestions. `http://www.gamepix.com/blog/the-big-list-of-2d-html5-games-engines/`. Accessed: 2016-03-02.

[34] Scirra forums. `https://www.scirra.com/forum/`. Accessed: 2016-03-02.

[35] Scirra: Construct 2 multiplayer features. `https://www.scirra.com/manual/174/multiplayer`. Accessed: 2016-03-02.

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

[37] Unity supported features. `http://docs.unity3d.com/ScriptReference/index.html`. Accessed: 2016-03-02.

[38] Unity networking feature support. `http://docs.unity3d.com/ScriptReference/Network.html`. Accessed: 2016-03-02.

[39] Websocket-sharp plugin for unity. `https://github.com/sta/websocket-sharp`. Accessed: 2016-03-02.

[40] Unity statistics and public relations. `https://unity3d.com/public-relations`. Accessed: 2016-03-02.

[41] Unity forums. `http://forum.unity3d.com/forums`. Accessed: 2016-03-02.

[42] Unity 2d development forum section. `http://forum.unity3d.com/forums/2d.53/`. Accessed: 2016-03-02.

[43] Suggestion: Gamemaker instead of unity. `http://forum.unity3d.com/threads/where-to-start-2d-toolkit-or-platformer-pro.380770/`. Accessed: 2016-03-02.

[44] Github, version control system. `https://github.com/`. Accessed: 2016-03-02.

[45] socket.io.js documentation. `https://docs.omniref.com/js/npm/socket.io/0.8.4`. Accessed: 2016-03-02.

[46] Average household network bandwidth in the united kingdom. `http://www.ispreview.co.uk/index.php/2015/02/`

`ofcom-average-uk-home-broadband-speeds-slowly-reach-23mbps.`
`html`. Accessed: 2016-03-02.

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

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

[49] 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.

[50] 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.

[51] 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.

[52] 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.

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

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

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

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

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

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

[59] 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].