A Network Extension for GameMaker Browser Games

Teun Kokke

Honours Project Dissertation
Software Engineering
School of Informatics
University of Edinburgh

2016

Abstract

GameMaker by YoYoGames is a game development tool that in 2011 released a feature to export its games to the HTML5 web platform. This HTML5-export feature is however incompatible with the networking capabilities that do work on other export platforms supported by GameMaker. This constraint causes developers to use less well-suited browser-game development software that do support networking features. In this paper, we therefore develop a networking extension for GameMaker that can be used by the average (novice) developer in the GameMaker community.

The extension is written in JavaScript, and uses Socket.io to handle interactions through the network through a TCP connection. This extension comes along with a matching client and server, which are set-up to act as guidance for other developers. The client is a GameMaker game, written in GML (GameMaker Language), that makes use of the extension by mapping the extension's functions to functions in GML. The server is written using Node.js in combination with Socket.io, providing reliable communication that remains unobstructed by NAT devices.

We provide insight into how each of the three components are written and can be extended by other developers. The system is evaluated in terms of scalability and performance. The evaluation results indicate that the system performs well by matching it with network properties of other professional multiplayer games. Concurrent connections cause the server performance to scale linearly. The average RTT (Round-Trip Time) remains unaffected by concurrent connections up to 8,500 incoming messages per second, after which it rises beyond 100 milliseconds per packet on average. Client distance also affects the RTT, but only becomes a bottleneck when approaching high 1,000s of kilometers from the server.

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 assistance, feedback and ideas without which this project would not have been the same.

List of Acronyms

- **GML** - GameMaker Language: the primary language used in GameMaker. Its syntax is similar to those in the well-known C based language family such as C++, C# and Java [1].

- **API** - Application Program Interface: a set of programming instructions and standards for accessing a Web-based software application or web tool [2].

- **IP** - Internet Protocol: the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries [3].

- **TCP** - Transmission Control Protocol: a core protocol of the Internet protocol suite, providing reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network [4].

- **UDP** - User Datagram Protocol: a transport protocol that uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network protocol. There is no guarantee of delivery, ordering, or duplicate protection [5].

- **W3C** - World Wide Web Consortium: an international community that develops open standards to ensure the long-term growth of the Web [6].

- **P2P** - Peer-to-Peer: a distributed application architecture that partitions tasks or work loads between peers. Peers are equally privileged, equipotent participants in the application [7].

- **RTT** - Round-Trip Time: the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received. This time delay therefore consists of the propagation times between the two points of a signal [8].

- **RSS** - Resident Set Size: the portion of the process's memory held in RAM [9].

- **CPU** - Central Processing Unit: the processor of a computer; an electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions [10].

- **RAM** - Random Access Memory: a form of computer data storage. A random-access memory device allows data items to be accessed (read or written) in almost the same amount of time irrespective of the physical location of data inside the memory [11].

- **LAN** - Local Area Network: a computer network that interconnects computers within a limited area such as a residence, school, laboratory, or office building [12].

- **GUI** - Graphical User Interface: a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation [13].

- **IDE** - Integrated Development Environment: a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger [14].

- **IETF** - The Internet Engineering Task Force: the body that defines standard Internet operating protocols such as TCP/IP [15].

- **D&D** - Drag and Drop: a pointing device gesture in which the user selects a virtual object by "grabbing" it and dragging it to a different location or onto another virtual object. In general, it can be used to invoke many kinds of actions, or create various types of associations between two abstract objects [16].

Table of Contents

Chapter 1

Introduction

In the current day and age, many people spend a large portion of their time playing browser games. These games are appealing as they 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 [17]. They are highly platform independent and potential users are generally easy to reach. Also their updates are seamless and can be implemented without requiring repeated patch downloads to a hard drive.

The market in this area has therefore been booming. Developers 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 new games can be released.

All of the above reasons trigger rapid advancements in the world of browser game technologies and development. In background chapter 2, we describe how the newly released web standard, known as HTML5, allows game graphics and engines to be executed directly inside the web browser without the need of third-party applets, how some game development tools (including GameMaker by YoYoGames) have therefore been expanded with the capability to support the development of games that are compatible with HTML5, and find GameMaker's absense of browser-compatible networking features.

This incompatibility is forcing a limitation onto its games, and causes developers to instead opt for less well-suited browser-game development software that do support networking features. We therefore compare a variety of alternative and related tools in chapter 3 that can be used to create these games, and discover that GameMaker remains one of the better tools for novice developers to develop 2D games in a short period of time, given that it would support networking capabilities. For this reason, we create an extension to GameMaker's functionality that does support networking features on the HTML5 platform.

We find minimal requirements for the extension to provide by making a comparison to professional multiplayer games, which allow a throughput of at most 150 milliseconds and suggest it to be below 100 milliseconds for an optimal connection. They are able to host several hundreds, to a few thousand concurrent and interacting connections on a single server.

We provide information regarding different data transportation mechanisms and protocols, find different technologies (Socket.io and WebRTC) that implement these protocols, briefly introduce a relatively modern server-side technology, Node.js, and cover quality attributes that guide the development decisions at the design and implementation stage of this thesis.

We then demonstrate how the extension is created and can be added to GameMaker in the development chapter, 4. We also design a game client using GameMaker that implements this extension, and a server system with which the client can connect. Both the functional and non-functional requirements for each of these three components are specified, explaining what their tasks are, how these are executed, which of the quality attributes are considered and how these are actualised into complete implementations for multiplayer GameMaker browser games.

The implementation provides an insight on how the components in the system are connected. This is important as any novice GameMaker developer with little-to-no experience in networks should be able to utilize, modify and extend the provided set-up, and create their own multiplayer 2D browser game using the extension.

The functions provided by the extension are made to be easily-translatable from those natively supported by GameMaker. Therefore, also multiplayer games developed with GameMaker that did previously not work in the browser can be translated into a browser-compatible version within a short amount of time.

The system's implementation as a whole is then evaluated in chapter 5, using the benchmarker application, by inspecting the server's behaviour when loaded with certain levels of stress, as well as the impact this has on the round-trip times that are observed at the application level in each of the clients.

In attempt to find out when the system fails at maintaining the round-trip times below the maximum threshold of 100ms for an average server load, we find that the distance to the server is the most significant bottleneck. With a server located in Edinburgh (UK), clients located in India and South Africa[1] are in the gray zone where the throughput could affect the gameplay (between

---

[1]Generally countries outside of Europe

100ms and 150ms). Clients located in California (USA) and Thailand should logically be dis-allowed from participating on the server if we complied to the ping-limitation rules in professional multiplayer games. They cause a level of unfairness which breaches the maximum accepted threshold, with a roundtrip time of 150 milliseconds, impairing their own experience and those of others.

In the closing remark, chapter 6, we clarify that we cannot offer a reasonable solution to the distance problem, other than to decrease this distance by distributing servers across the world. The method is however arguable as it causes complex situations around synchronizing the individual servers. These synchronization issues beat the purpose of providing a simple-to-implement extension for the "novice developers who wish to create browser games at a fast rate".

We also briefly discuss optional future improvements and how the extension does not guarantee novice developers from intuitively understanding how the system works. For this matter, it would be valuable to receive further feedback in terms of usability from these developers directly.

Chapter 2

Background

2.1 Advances in Browser Game Development

The following section introduces new features that arise due to advances in web browser technologies, and how these advances contribute to a new era in game development. We will explain what GameMaker is, how it uses these new technologies, and point out its current limitation in terms of creating networked applications for browser games.

2.1.1 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 [18].

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 now either deprecated or entirely unsupported by most devices [19].

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" [20]. The element can thus be used to draw graphics in JavaScript with the "Canvas API" [21].

---

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

5

2.1.2 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" [22].

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 [23]. 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 [22].

The main GameMaker community forum is hosted directly by YoYoGames [24], 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 [25] which hosts an additional 557,000 posts for 56,000 topics for almost 18,000 members. 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 [26].

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 [27], YoYoGames has never included this feature [28].

An attempt, and in fact a well established blueprint for the implementation of this thesis under the name "39js", was developed by a member of the YoYoGames community [29]. It is however a discontinued project, considered to be in alpha stage and fails to be used by certain developers as the setup contains errors and provides no guidance towards a fix. It is a basic setup that attempts to broadcast messages as they come in without providing any further underlying structure.

2.2 Network Expectations

Applications often have different demands from the network. One must therefore consider the technical aspect of the network. For example, fairness in a game is crucial for an enjoyable gameplay. This is especially true for games containing elements where speed or response time is important [30]. Servers in modern high-end multiplayer games are typically loaded from several hundreds, to a few thousand simultaneously connected clients, depending on the

level of interaction between the clients.

On various games, clients may automatically get disconnected from the server when their ping or round-trip time (RTT) reaches beyond a certain threshold as these clients potentially disrupt the gameplay. This threshold can vary depending on the game. Fast-paced games such as Call of Duty and Battlefield tend to have this threshold set around 150ms to 200ms.

2.3 Data Transportation Mechanisms

The following section covers underlying knowledge in terms of the transport protocols TCP (Transmission Control Protocol) and UDP (User Datagram Protocol), as well as arising issues due to NAT (Network Address Translation) traversal.

2.3.1 TCP and UDP

Communication protocols are the foundation of the network layer, and play a big part in colouring the behaviour of the network speed and quality.

TCP is a reliable connection-oriented and error-free process-to-process data transmission protocol for interaction between different hosts. After establishing a connection using an event referred to as the "three way handshake" [31], it keeps the connection open [32]. It automatically takes care of 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 the added reliability adds overhead [33], making the packets are relatively large in comparison to UDP.

UDP is a connectionless transmission protocol that, unlike TCP, provides a "minimal, unreliable, best-effort message-passing transport to applications" [34]. It provides no guarantees for delivery nor delivery order and offers no protection from duplication. Despite these clear down-sides, due to the relatively smaller header size, it may often be useful for applications that value the speed of the connection above reliability of the content. Developers can namely implement their own reliability-handling mechanisms in the application that are optimized to the needs of their system.

2.3.2   NAT traversal

Due to the limitation of IPv4 addresses, novice game developers are often required to deal with NAT traversal [35], as they start off hosting their game on their home computer. This computer is typically part of a private network[2]. Network address translation (NAT) is a mechanism that takes care of assigning private IP addresses to local devices within the network, whilst maintaining the same global IP to identify the local network as a whole.

This issue adds the requirement for server owners to set-up port-forwarding in their router. This is the act of specifying which device within the local network the router should forward incoming messages to, and this decision is based on the port number that has been assigned.

As UDP does not maintain a 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 [36].

2.4   Networking at the Application Level

The following section covers several technologies that can be used by developers to create network applications.

In a typical multi-user game scenario, game clients are described by the server as a set of parameters. These parameters represent the "game state". Clients keep pinging and updating the server with small messages containing their own state[3], asynchronously[4]. Fairness is diminished when the clients' and server's game state become desynchronised [30], which often negatively effects the user experience of the game.

2.4.1   WebSocket

WebSocket is a protocol that "provides a method to push messages from client to server efficiently and with a simple syntax" [37]. It was standardized by the IETF (Internet Engineering Task Force) 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" [38]. 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]A group of devices that are mapped to the same global IP address
[3]Represented as a set of variables
[4]without blocking the browser, allowing users to perform operations uninterruptedly

2.4.2 Socket.io

Socket.io is an event-driven JavaScript library that can be used both on the client's browser, and a server [39]. It supports features that allow sending and receiving data using the WebSocket protocol, without interruption of the code flow [40, 41]. For this reason, it is **often used in combination with Node.js**.

2.4.3 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" [42]. Although certain web applications have already been created with WebRTC (mainly for P2P video streaming [43, 44]), no proper standards are out yet [17].

This, combined with the 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 [43], makes UDP many times **more complicated** for developers to handle efficiently.

2.5 Node.js

Traditionally, servers in a network create a separate thread for each client, therefore rapidly running out of RAM and keeping clients on hold until sufficient memory for a new thread is released [45].

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" [45]. 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 [46, 47], as opposed to the traditional, less scalable servers.

2.6   Quality Attributes

Despite having considered the basics of some underlying functional requirements and technologies, there are also other aspects that need to be considered when developing a piece of software, namely the "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 it is designed. Definitions to some quality attributes that are especially important to this project and will be referred to in the future are listed in table 2.1 below.

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 [48].

Chapter 3

Related Tools

In this chapter, we identify competitors to GameMaker. We discuss how, despite offering many valuable features, GameMaker remains one of the better tools for novice developers that wish to create 2D browser games at a fast pace.

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 development tools that also natively support networking features. 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, we 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 [49] represents that of the Google Play Store [50] and the Apple App Store [51], and may therefore be considered to be a reasonable method for finding the more commonly used tools.

developer.mozilla.org [52] (world rank 198 by alexa.com [53]) 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[1] [54] hosts a list of similar tools, but is less detailed about their sources.

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

---

[1]Hosted at gamepix.com

Tool	Score	Voters	Mozilla[2]	Gamepix[3]	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

3.1 Construct 2

Construct 2 is shown to be a direct contender. It provides its own GUI and pursues similar goals as those of GameMaker: simplifying the development for novice programmers. This is done by supporting inbuilt D&D (Drag-and-Drop) 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 [55].

In April 2014, release "r164" to "r168" updated the engine with networking features by allowing users to create a said "network object" [56]. 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" [57].

There are many methods of establishing a network between processes on different hosts, and managing the trade-offs in terms of speed and reliability. For the act of simplifying the game development and demands in terms of under-

---

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

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

standing "how multiplayer online games fundamentally work" of the developer, we want to provide a black box[4] that sufficiently handles both[5].

3.2 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 [58].

Networking features are supported as a D&D class [59], but can also be used by installing the "WebSocket-Sharp" plugin in C# [60].

Currently there are roughly 4,500,000 users registered [61] to the Unity forum [62], however only a small percentage of this is active in 2D development as this only hosts 5,600 topics [63].

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?" [64] are therefore a common response.

3.3 Other tools

Pixi.js, EaselJS, Crafty.js, Phaser and Quintus are all legitimate tools for developing games. They are however merely JavaScript frameworks and libraries that assist developers to create 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).

---

[4]"a system which can be viewed in terms of its inputs and outputs, without any knowledge of its internal workings." - `https://en.wikipedia.org/wiki/Black_box`

[5]i.e. ensured packet delivery and providing round-trip times below 100ms for at least a few 100s of concurrently active clients

Tool	Topics	Community main page
Pixi.js	1455	www.html5gamedevs.com/forum/15-pixijs/
EaselJS	738	stackoverflow.com/questions/tagged/createjs[6]
Quintus	n/a	plus.google.com/communities/104292074755089084725
Crafty.js	1666	groups.google.com/forum/#!forum/craftyjs
Phaser	8793	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

This chapter describes the development process of three components: the extension, a matching client, and the server. The chapter gives a brief outline of both functional and non-functional requirements of each of these components, as well as the decisions that were made to ensure these requirements are met.

We then dive deeper into the actual implementation details, with the aim to teach the reader how the extension can be utilised, modified and extended in order to create his/her own multiplayer browser game using GameMaker [65]. Interaction patterns and sample code snippets are therefore provided for each of the components, and working applications will be showcased.

4.1   Prior Considerations and Design Decisions

4.1.1   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 be executed 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 [66].

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.

4.1.2   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 [67].

The extension: Performance is an important aspect for the extension, without which user satisfaction could be decreased. For this reason, we will measure how well the extension performs in terms of the networking capabilities it needs to provide. It is not expected to be modified by any developer, as it already supports all features required to make a live networked application run. Reliability is also an important attribute, as when the extension fails to transmit or receive messages, the developer would have to learn how to handle these situations. 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. Modifying a game client often means that the server needs to be modified as well, in order to keep it matched with the client set-up. The developer should

therefore 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 even needs to be scalable as the server performance may decrease when large numbers of clients may wish to be connected, which implicitly deteriorates the user experience.

Also security becomes an important aspect of the server as it may receive unintended or malicious requests. We do however not consider this to be of primary importance, and developers may prefer to implement their own security 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.

4.1.3 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). We therefore choose to use JavaScript in combination with Node.js on the server-side.

Socket.io versus WebRTC: The behaviour of the network is affected by the transmission protocols that are used to run it. Developers with little to no experience with networks might not know how to deal with certain kinds of network behaviour. Choosing the right system that offers predictable behaviour to take care of the data transmissions is therefore crucial.

Generally speaking, TCP is a form of reliable data transmission. But this added reliability causes extra overhead on the network, and makes it known to be slightly slower than UDP when the network has only 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 a conclusion 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 general flow of interaction in the server is displayed in Figure 4.1. Each component aims to be easily maintained by handling only a single responsibility. The responsibilities are clarified directly in the naming of the variables and methods which contributes to the conceptual integrity of the code, as indicated later in Figure 4.4.
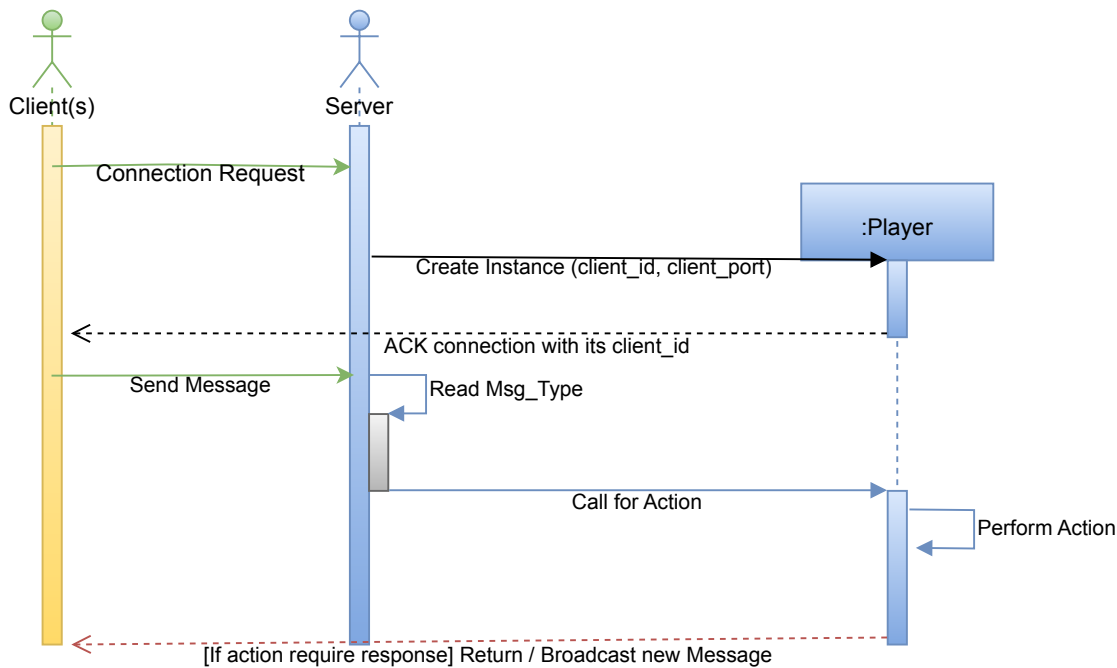

Figure 4.1: Sequence diagram displaying the overall server logic. Note that clients can asynchronously and repeatedly send messages to the server after having received their server acknowledgement (ACK) packet. The `Msg_Type` can be anything such as `chat_message`, `update_user` or `client_disconnected`, and the action taken by the server is specified by the player object method, referenced by the action dictionary on the server.

The precise order of the server logic is explained as follows:

1. Upon receiving a connection request, the server instantiates a client in which initially the client's socket and client_id are stored. The socket will be connected directly to the client machine that sent the connection request (Figure 4.2, 1).

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

3. If at any time the server receives a message from this (or any known) client, the server will read the first line from the buffer containing the purpose of this message and store it as message_code (e.g. ping, send-message, update-user) (Figure 4.2, 3).

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

5. Depending on the action (Figure 4.3, 5), the client instance will receive a function call that handles the message. The exact list of actions is specified in `messages.js`, as well as in the client as follows:

```
{
        "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
        }
}
```

The client-representing 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 (i.e. stored in the Player instance). Another benefit to this is that the client is allowed to send smaller, more specific messages.

```
/* 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).

```
/* 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: Dictionary storing the high-level actions for a specified player instance to perform.

The code on the server acts as a template, 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 with new functionality depending on any specific need the developer may have.
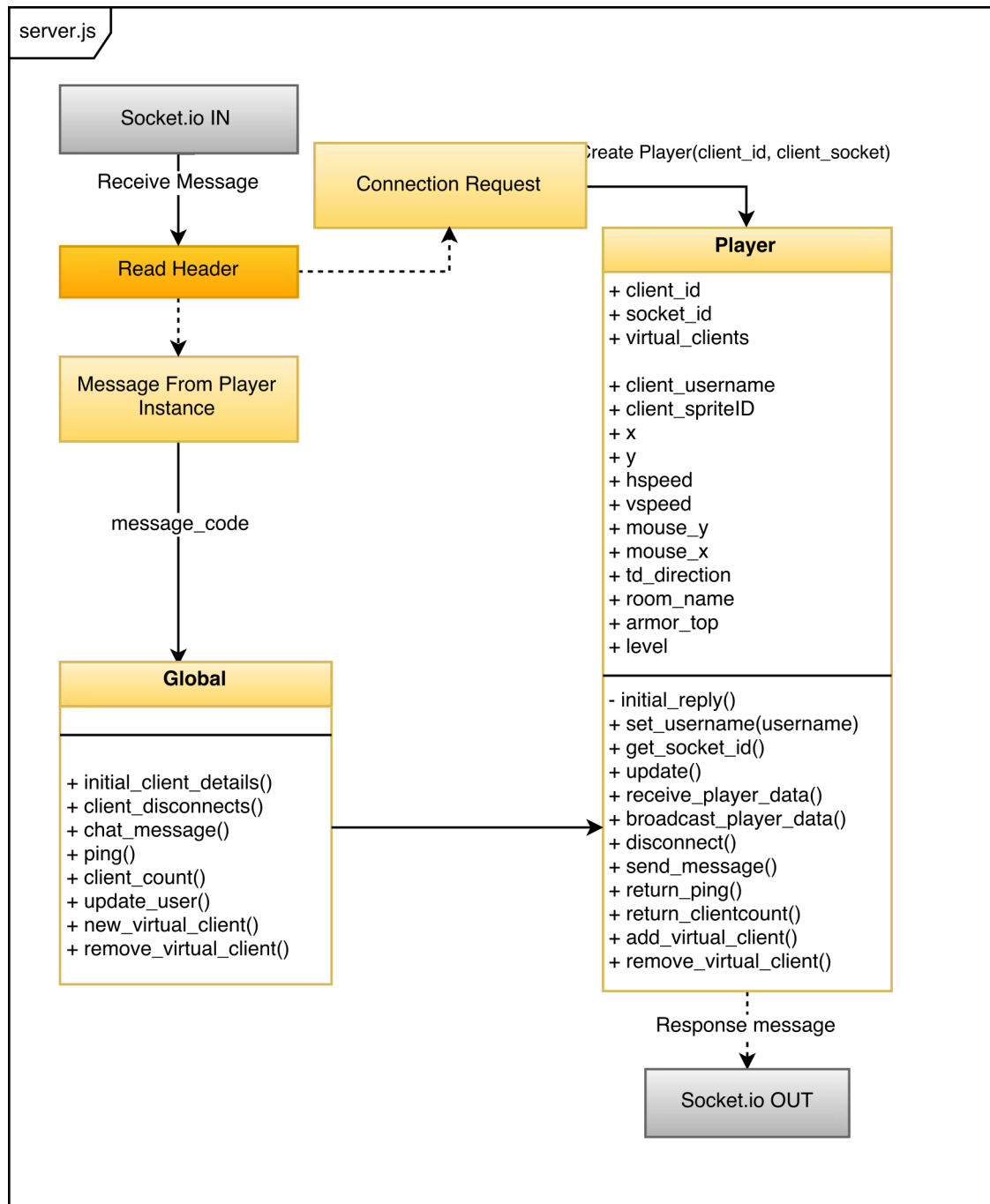

Figure 4.4: Class diagram indicating the interaction between incoming messages through Socket.io, and the methods provided by the Player class that are used to handle these messages.

```
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.
    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: Indicating the close relation between the Player class on the server matching the Player object of the GameMaker client. Upon instantiation, the constructor automatically returns a message to the referenced client with it's unique identity defined by the server.

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   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 essentially a set of JavaScript functions that can be called externally from inside the GameMaker GUI in GameMaker's native programming language "GameMaker Language"

(GML). These functions in turn provide the networking functionality. The extension consists of three components:

1. An MIT-licensed opensource socket.io JavaScript library provided by Learn-Boost: socket.io.js (documentation ref. [68])

2. A set of networking functions, located in GMWebNet.js

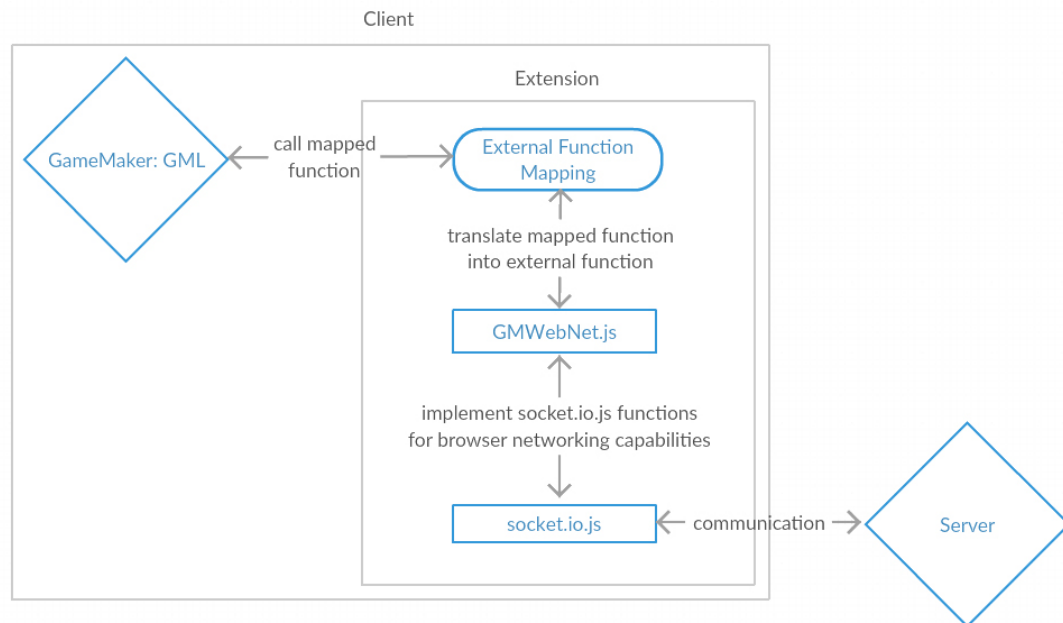3. A mapping from GML functions to the external networking functions

Figure 4.6: The extension flow chart, indicating where the components are located. The client executes in the user's browser. The mapper provides functions in GML which corresponds to the external functions located in GMWebNet.js with a matching number of arguments and argument types. These external functions are closely coupled with socket.io.js, which is responsible for the interaction from and to the server.

socket.io.js   is used for providing the core networking functionality, transmitting and receiving packets as it supports the most important quality attributes for typical applications developed with GameMaker. It offers underlying reliability and stability by using TCP to transmit the messages and guarantees a successful traversal through NAT devices without demanding additional attention from the developer.

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 creation, memory buffer and incoming messages. Developers do not

need to modify this file. It is considered to be a black box which provides the networking foundation on top of which networked applications 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 client matching that socket is disconnecting. The server interprets 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 function `buffer_write(value)`.

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.

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.

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. Functions with the following purposes are included:

- initiating a connection
- creating a packet in the buffer
- sending a packet
- spotting incoming messages
- reading incoming messages
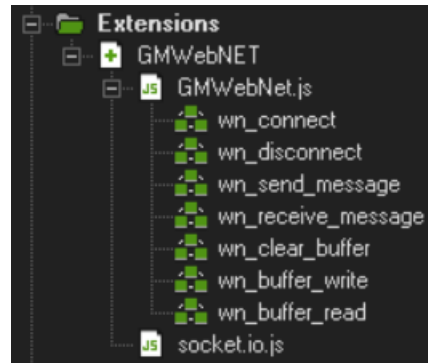- disconnecting hosts

Figure 4.7: View inside the GUI that visualises the extension function mapping setup after correctly importing the extension into GameMaker.

```
<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)

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).

4.2.3 Client

In order for the networking features to work with any GameMaker client, one simply has to import the extension `GMWebNet_extension.gmez`. This can be done by `right-clicking the extensions folder` ⇒ `Import Extension`. After selecting the extension file, all networking features will be automatically set-up and ready for use.

Exact implementation details can be found in the included GameMaker application files by opening the corresponding project files in GameMaker Studio:

1. Benchmarker project:
   `/client_benchmarker/benchmark.gmx/benchmark.project.gmx`

2. Chatroom project:
   `/client_messenger/client.gmx/client.project.gmx`

Any object can create and send messages to the server using the functions listed in table 4.1. For example, the Player instance can send its data to the server as follows[1]:

```
//clear the buffer content to prepare a new packet
wn_buffer_clear();

//add the message type
wn_buffer_write(wn_msgtype("C2S", "update_user"));

//store the player variables in the send buffer
wn_buffer_write(username);
wn_buffer_write(sprite_id);
wn_buffer_write(x);
wn_buffer_write(y);
wn_buffer_write(h_speed);
wn_buffer_write(v_speed);
wn_buffer_write(mouse_x);
wn_buffer_write(mouse_y);
wn_buffer_write(roomname);
wn_buffer_write(level);

//send the packet to the server
wn_send_message(server_socket)
```

However for receiving packets, it is considered to be a good practise to create a special listener object (e.g. `obj_Network_Controller`), which is generalized to take care of the single purpose of receiving incoming messages from the network. This object can then appropriately update other instances with the newly received game state from the server. By managing the single responsibility of this object, we ensure that incoming packets are gathered and collected

---

[1]taken from the chatroom project

at a single centralized location. It is therefore easier for other objects to know where to access this data from, increasing the managability of the code.

After initializing the connection using the function `wn_connect(ip_address, port)` provided by the extension, one only has to set-up a socket-listener loop which will respond as soon as a new message arrives. This is done using the function `wn_receive_message(socket)`, and when it returns a non-zero value, the incoming message can be read from the buffer line-by-line using the function `wn_msgtype("S2C",wn_buffer_read())`.

An example of how this can be implemented is as follows[2]:

```
//listen for incoming messages from the server
while(wn_receive_message(server_socket))
{
  //read the type / purpose of the message
  var msg_type=wn_msgtype("S2C",wn_buffer_read());

  switch(msg_type)
  {
    case "chat_message":
      //receive message from some user
      var sender        = wn_buffer_read();
      var chat_message  = wn_buffer_read();

      //add new message to the chatbox
      obj_Chatbox.chat_contents = add_line(sender, chat_message);

      break;
  }
}
```

After creating a game-like mechanism the application can be exported to the HTML5 target module with GameMaker, which automatically converts the project into a browser-executable javascript file. The extension will be integrated with this file, and continue to take care of handling the network.

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.

---

[2]taken from the chatroom project

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 in Figure 4.9 the global data can be found. 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 above 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.
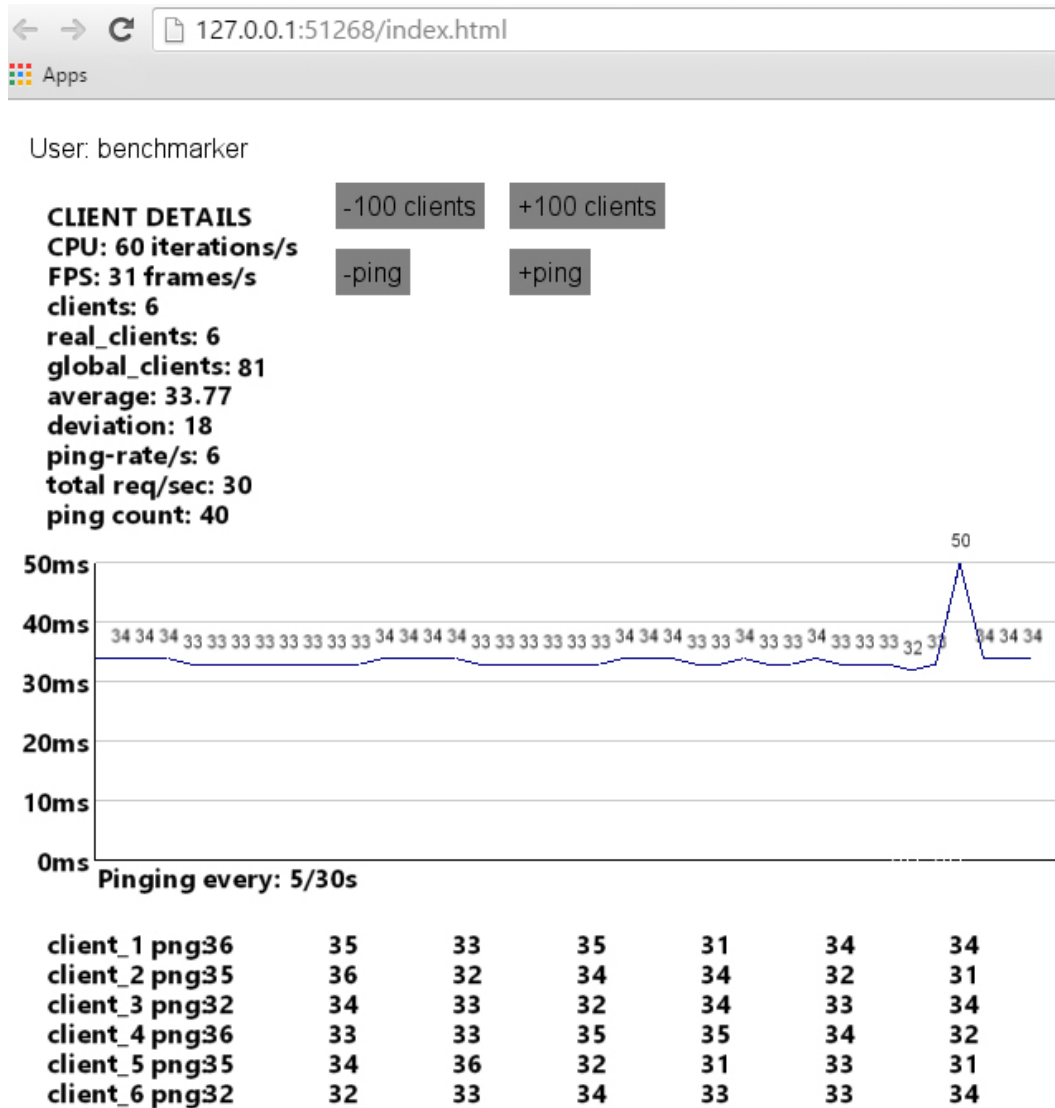

Figure 4.9: Displaying the GUI of the benchmarking application. On the left top the global data is available for the client. The graph below displays the average RTT over time for these clients. The table on the bottom displays the virtual clients and 7 of their most recently observed round-trip values.

4.3.2   Chatroom Application

Another application was created, both for demonstration and template purposes to other developers. This application is in the form of a chat room where users can log-in, walk across a room and broadcast messages to each other. A screenshot of this application is provided in Figure 4.10.

Upon startup, an instance of the game-controller object (`obj_Controller`) is

instantiated which prompts the user for a username. Once this username is set, it instantiates the network-controller (`obj_Network_controller`) and the player instance (`obj_Player`).

The network controller then initiates a connection with the server using the function `wn_connect(server_ip, server_port)`. It then awaits incoming messages from the server. The network controller is also responsible for forwarding user data such as user coordinates, cursor position etc. This is all done using the functions described in Table 4.1.

Figure 4.10: Ingame screenshot of the messenger application, showing three players connected on localhost, and interacting with eachother through the message box in the bottom-left corner.

Chapter 5

Network Extension Evaluation

In this chapter, the extension is evaluated in terms of fairness and performance. More specifically, we test how concurrent connections impact the server performance in terms of CPU (Central Processing Unit) usage, CPU processing time and RSS (Resident Set Size). We then observe how the server performs and scales based on client interactions, discover how broadcasting messages[1] does not affect their round-trip time, and that although large distances do affect the round-trip time, countries located within Europe maintain a round-trip time well within the acceptable 100 milliseconds.

5.1 Controlled Network

Controlled experiments were conducted in order allow making predictions about the server's behaviour when loaded under similar stress in future occasions.

These experiments were performed 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 [69]. 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 and how well it scales before it runs out of resources such as memory and

---

[1]as opposed to one-to-one interaction

processing power. 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.

The setup   includes a variable number of concurrent connections consisting of up to 15 physical instances with each simulating at most 500 clients. These clients do not contact the server after establishing a connection. The resulting values are averaged over 2-minute run duration periods, and each after 2-minute period the experiment restarts, having the number of clients that attempt to connect simultaneously increased by 100.
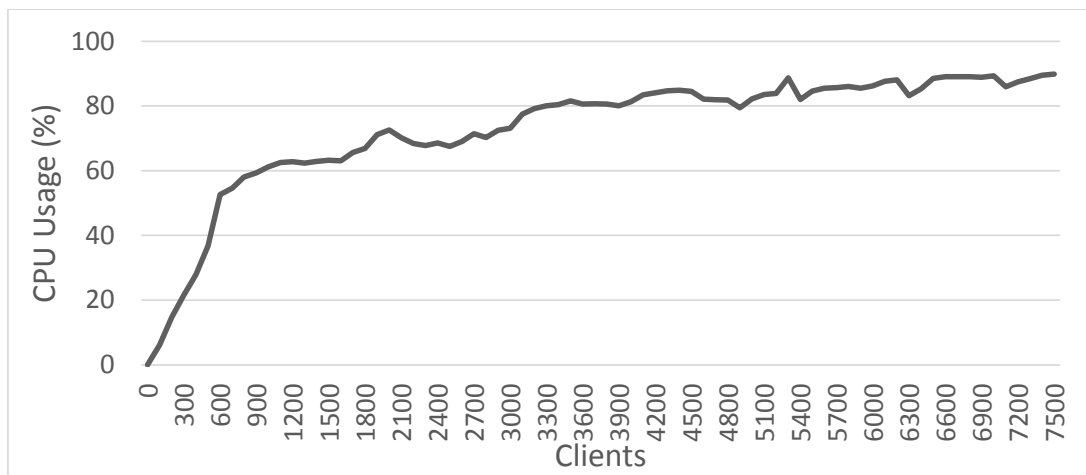

Figure 5.1: Displaying the CPU usage for the server process handling a variable number of client connection attempts in percent.
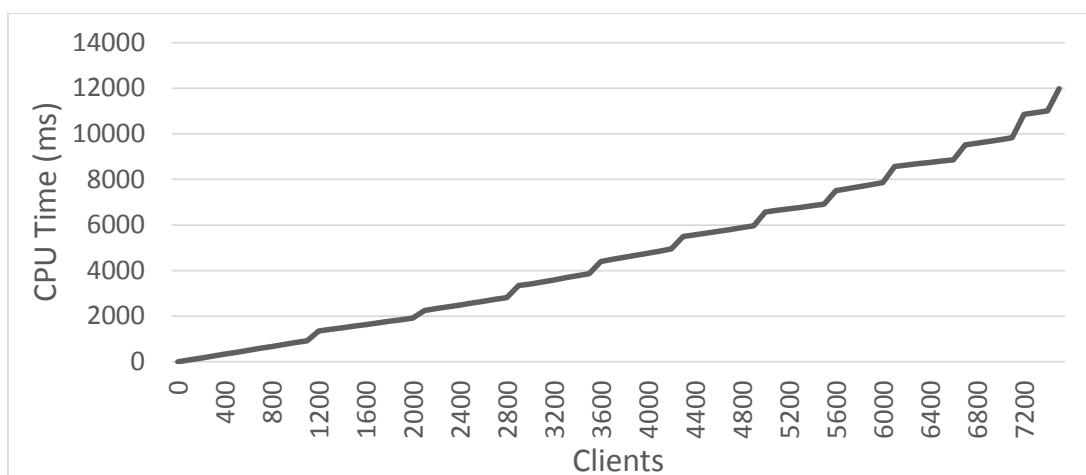

Figure 5.2: CPU time in milliseconds, displaying the amount of time required for the server to process the connection requests of all clients.
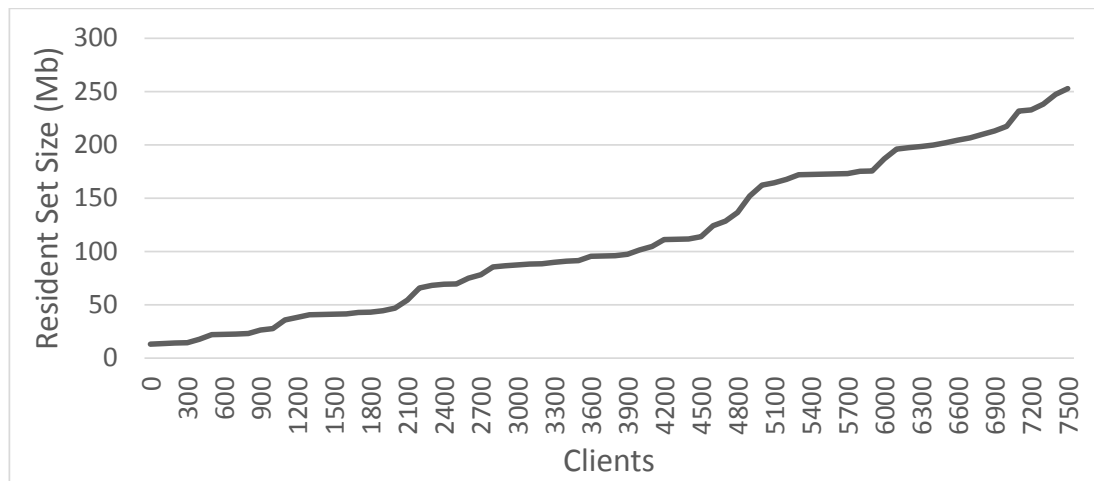
Figure 5.3: Resident set size in Megabit, showing the portion of RAM that is occupied by the server process.

Figure 5.1 indicates a logarithmic trend in the CPU usage on the server. This behaviour appeared strange at first. Although this effect is explained by taking the CPU time in Figure 5.2 into consideration.

The system aims to run the process at an optimal CPU usage amount, which can be noticed by the sudden change at a CPU usage of roughly 60%. As soon as the server's processor reaches this point, it will attempt to balance the usage by allowing more processing time for connecting the clients. The connection request processing time scales in a near-linear fashion, indicating that connecting a new client happens in constant time almost regardless of the amount of already-connected clients on the server. Occasional peaks and the slight increase in the trend of the CPU time may indicate the processor demanding extra time to process the increased number of connecting clients.

Due to the connection handling mechanism of Node.js, where each newly connected client is offloaded to a heap using constant memory space, the RSS scales up in an almost linear fashion. This property makes it easy to predict the amount of memory required for a certain expected load. For example, in the observed scenario, each client consumes roughly 5KB of the RAM. Therefore when expecting 10,000 clients to connect to the server, they will be using up around 50MB (slightly higher due to the trend not being exactly linear). Similar experiments have found similar behaviour, where 16GB is consumed to record a total of 1 million concurrent connections [70].

5.1.2 Message Broadcasting Performance

The concurrent connections experiments have indicated that the system scales well within the expected demands of an average multiplayer game in terms of client concurrency. The following experiment evaluates what happens when these clients start interacting. It observes 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.

The setup   involves up to 5000 concurrently connected clients (10 physical instances connected by a cable, simulating at most 500 clients each) all located within the same local network. Clients can send 8-Byte packets, and each client sends these packets at regular time intervals. Each client measures the round-trip time of the packet, and for each interval these round-trip times are averaged.

The values in Figure 5.4 were generated by combining several experiments, each containing a different fixed number of clients: 50, 100, 500, 1000, 2500 and 5000. In these experiments we modified the rate at which each client transmits messages to the server (a higher sending rate results in more incoming messages per second at the server).

RTTs of over 200ms would significantly negatively impact the user experience. Each time when the average RTT exceeds this threshold, the experiment is terminated. Moreover, beyond this point, the server appears to receive messages faster than it can handle, causing it to temporarily store the messages in a queue for future processing cycles. This behaviour results in a vicious cycle that causes the memory to eventually overflow, as clients keep sending packets at the same constant rate.
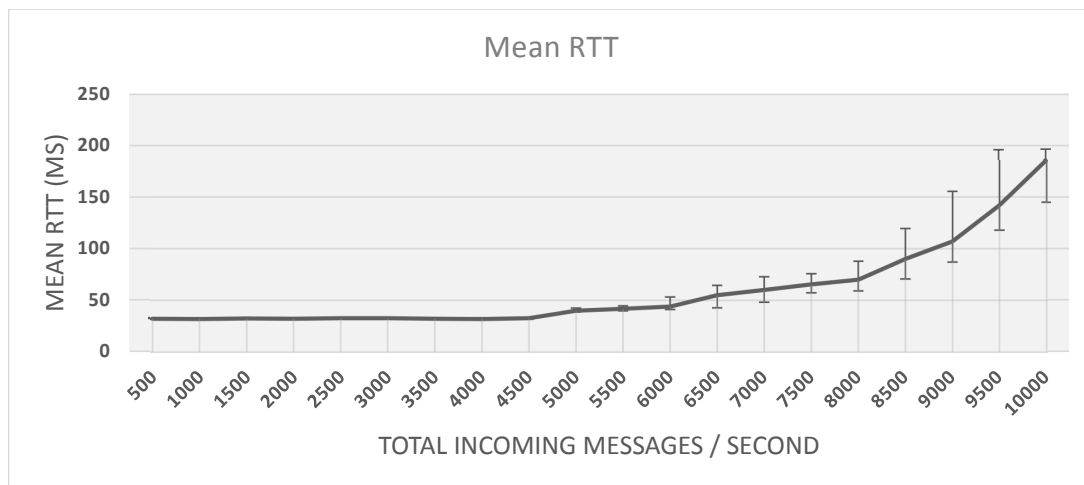

Figure 5.4: The general case scenario indicating the observed mean 8-byte message round-trip time, depending on the total number of messages incoming to the server at every second. Note that the final max-value is biased by the fact that some of the experiments were already terminated, as their RTT exceeded 200ms.

From Figure 5.4, it can be observed that the messages maintain a stable 32ms round-trip time until the server receives around 5000 messages per second. Beyond this point, the round-trip time grows exponentially until it reaches >200ms, which is when each of the experiments are terminated. A line can be drawn around 8000 to 8500 incoming messages per second (100ms RTT), as

this is the boundary where the user experience in professional fast-paced multiplayer games is considered to be affected. Note however that the real boundary depends largely on the expectation and possible delay-masking qualities of the application.

Note also that after receiving, these messages were broadcasted to all other clients (as opposed to one-to-one messages). In order to further visualise the effect of broadcasting messages from each client to every other client, the mean RTT results are displayed separately in Figure 5.5, as well as their corresponding standard deviation in Figure 5.6.
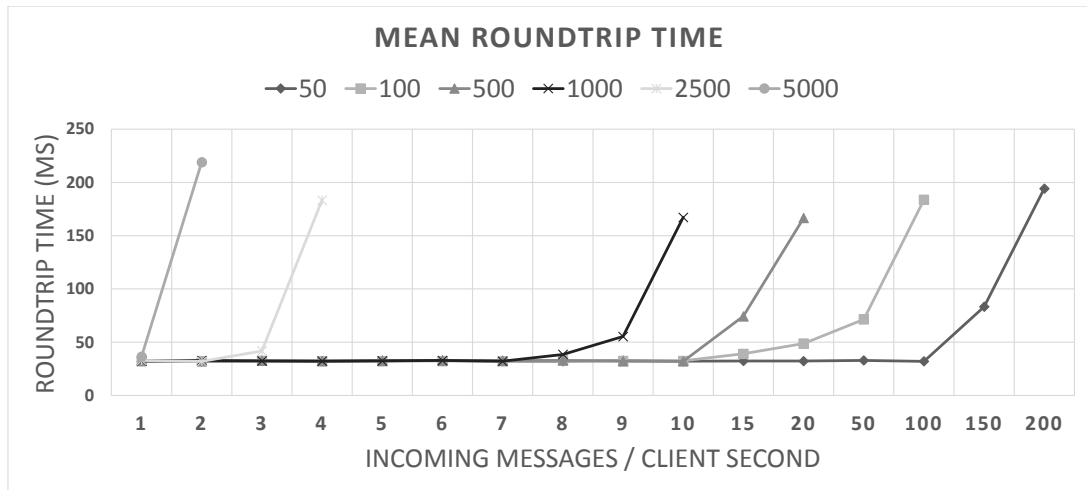

Figure 5.5: The mean round-trip time of all the messages that pass through the server, for different numbers of clients (50, 100, 500, 1000, 2500 and 5000). Each client sends messages to the server at variable rates.
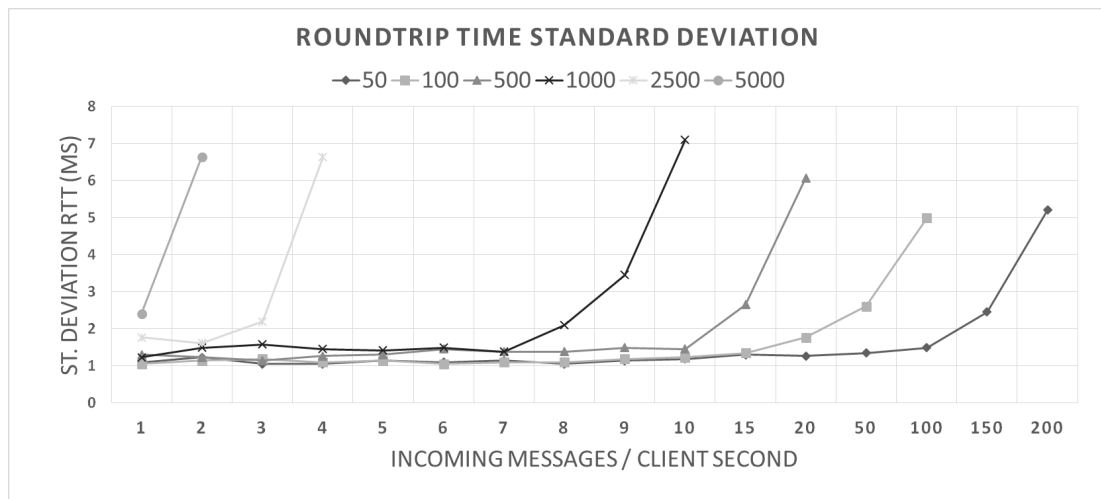

Figure 5.6: The standard deviation of the message round-trip times as displayed in Figure 5.5 for all messages that pass through the server, for different numbers of clients (50, 100, 500, 1000, 2500 and 5000) and for messages sent to the server at a variable rate.

The graphs in Figure 5.5 and Figure 5.6 indicate as expected that with few concurrently connected clients and the same send rate per client, the server manages to broadcast many more messages (as fewer clients will be sending messages). The number of messages the server needs to transmit every second is $sending\_clients \times send\_rate \times receiving\_clients$. It is important to remember that broadcasting is an intensive task with a complexity of $O(n^2)$ for n clients, as every client sends a message to every other client ($broadcasting\_clients == receiving\_clients$).

Therefore, if n = 100 clients are connected to the server, and each wants to broadcast 100 messages per second, the server receives 100×100 (= 10,000) messages per second, and has to send $100 \times 100 \times 100$ (=1,000,000) messages every second (although the sender technically does not need to re-receive its own message, this difference is negligible).

For n = 1000 clients, a maximum sending rate of 10 per second is observed. At this point the server receives 10×1000 (=10,000) messages per second, and has to send $10 \times 1000 \times 10$ (=100,000) messages per second.

These results are very interesting as they indicate that the average RTT is not necessarily affected by the number of messages the server has to send[2], but rather by the amount of messages the server has to receive and control in the message-handling logic.

5.2   Real Network Results

During the real network experiments, the network at the server was observed to be running at a consistent 54Mb/s download and 3Mb/s upload speed.

Multiple tests were executed with clients located at specified locations. In each case, the clients were again sending 8-byte messages to the server, at regular intervals of 1 second for 2 minutes. The 120 round-trip times for each client were then averaged in order to blur occasional peaks caused by inconsistencies in the network. The round-trip times of the clients in each common location were averaged, and the deviation of the round-trip times within the 2-minute time frame for 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, in order to prevent data transmission to affect the results, logs were saved locally at the clients. Only after each experiment was complete, these logs were forwarded for processing purposes. The round-trip times represent the delay times between the packet-send event in the browser client application, and the packet-receive event by the same client for the same packet.

---

[2]By considering that sending 100,000 messages results in a similar average message RTT as when sending 1,000,000, given that the amount of actually handled messages remains the same

5.2.1 Location-based Delay Fairness

The following experiments evaluate the effect of the geographical distance be-
tween 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, Cali-
   fornia (USA), India, Thailand, Germany, Hungary, United Kingdom and
   the Netherlands.

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

Table 5.1: Observed average round-trip times and the standard deviation per
test

Location	Average RTT	Server Distance
LAN	32ms	0km
Edinburgh	55ms	0km
Dundee	65ms	63km
Germany	67ms	977km
Glasgow	68ms	67km
France	69ms	1165km
the Netherlands	70ms	698km
Sweden	70ms	1238km
United Kingdom	71ms	66km
Hungary	76ms	1881km
India	103ms	7582km
South Africa	144ms	9760km
California (USA)	158ms	8065km
Thailand	171ms	9438km

Table 5.2: Average round-trip times per location, along with their geographical
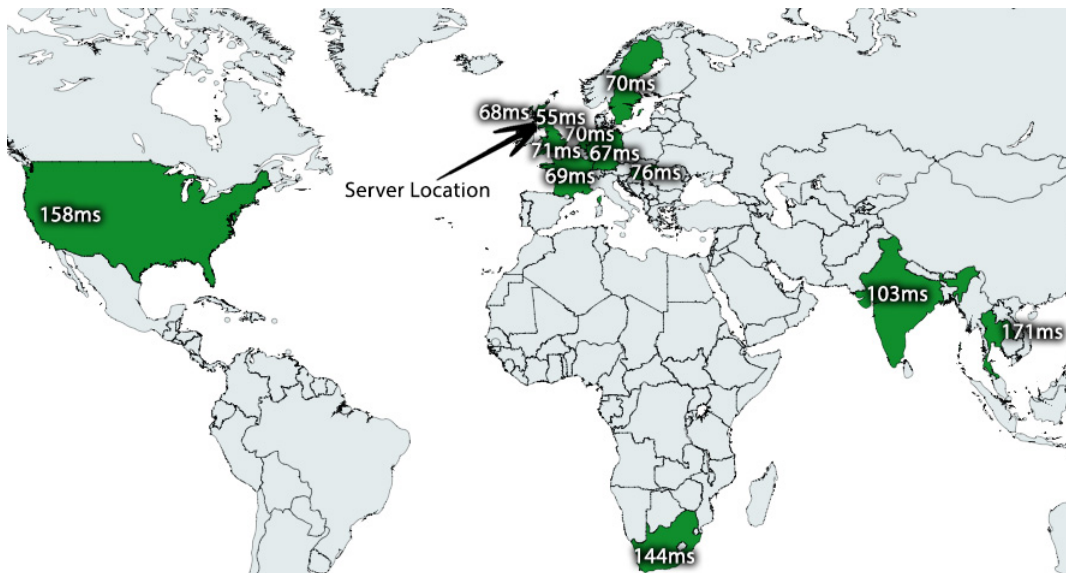distance from the server.

Figure 5.7: Worldmap displaying the tested locations and their according average recorded round-trip times

Clients located within the local network have the shortest round-trip time, as expected since this scenario requires packets to travel through the least number of nodes before reaching the endpoint. Clients located anywhere within Europe have an overall round-trip time between 55ms and 76ms. These values are reasonable depending on the purpose of the game that is created with the extension, but complicated to omit. The considered rule of thumb for RTT values is that anything below 100ms is acceptable, even for high-end shootergames such as Battlefield and Call of Duty, where clients may be disallowed from joining the server only when their round-trip time exceeds 150ms or 200ms for the reason of potentially disrupting the gameplay.

Table 5.1 and table 5.2 show the average round-trip times observed by clients located at different locations on earth. These RTT for these locations may differ due to factors i.a. the connection speed, number of traversed nodes, ISP quality and firewall settings. The experiment shows that the major contributor to unfairness in the network appears to be directly affected by the distance to the server, as the average RTT for each of the locations significantly increases as the distance increases. Figure 5.7 visualises the connection between the observed round-trip times and the corresponding distance between the client and the server.

Assuming that the added delay is caused by the implied distance and packet-switching, the only real solution to the problem involves modifying the physical link between the client and the server or locating the server closer to the intended clients.

Chapter 6

Closing Remark

6.1 Conclusion

The extension works well according to the specification in the functional requirements, and it can be implemented easily as its functionality is simpler than the native GameMaker networking features[1]. This is due to the fact that it automatically takes care of specifying the transport mechanism and the buffer along with its properties such as size, byte-alignment and type, taking away some of the control from the developer but narrowing down the choice he/she has to make about the network. Data types from the buffer are generic, matching with the weakly typed property of that of GML. Using Socket.io to open and maintain a TCP connection mitigates the NAT traversal problem that many novice developers would otherwise experience.

The extension offers sufficient scalability when combined with the provided server implementation using Node.js and Socket.io. This is proven by the fact that the server manages to maintain a few thousands of concurrent connections, and handles roughly 5000 messages per second over a long-time period before the packet round-trip time starts to visibly increase, with an accepted rate of 8500 messages per second before the round-trip time starts to exceed 100ms.

Long-distance connections are the main contributor to unfairness, but for this issue apart from distributing multiple servers across the globe or optimizing the physical link between the endpoints, no real solution is known. With a standard server load, any distance up to a few thousand kilometers is however found to provide an acceptable average round-trip time of less than 100ms.

Based on these experiments, we can claim that, in theory, developers can focus on creating their game without having to worry about designing and optimizing the work.

---

[1]that are supported for non-browser platforms

6.2 Discussion

The evaluation methods indicate that the network indeed performs well enough to allow a sufficient number of clients to connect and interact simultaneously without impacting a server with average-performing hardware requirements, proving that the extension is indeed useful.

However, proving that the extension is indeed usable to novice developers is a rather more subjective issue. Although the primary focus was to simplify the development process for other developers, this was approached by following the best "software engineering practices", enforcing the right quality attributes on each module of the implementation. A novice or even average browser game developer may not know about these practices. Therefore it would be fair to further evaluate the usability by inviting developers to try out the extension, and get a practical evaluation based on real user (developer) experience.

The time required to receiving accurate measurements forced this evaluation to be discontinued for this thesis project.

6.3 Future Improvements

Collecting feedback for a usability evaluation could indicate possible improvements in terms of simplifying the use of the extention to develop new networked browser games and applications.

Delay in round-trip times can be masked at the application level (either in the extension or in the client directly) by predicting future values based on recent value changes in the past.

Security features could be added to the extension, blending in with the provided black box set-up whilst ensuring authenticity between the client and server.

It may also be possible to optimize the network performance by using WebRTC to interact with the server. However this would only be beneficial GameMaker developers if also packet-losses and NAT traversal are resolved by the extension, and would not fix the increase in RTT caused by distance.

Fixing the distance issue can be done by spreading servers across the world, and connecting clients to the server that provides the lowest round-trip time. This, however forces multiple versions of the game state to coexist, thereby causing the issue where servers need to interact with each other in order to keep their versions of the game state synchronized.

Bibliography

[1] Definition of gml provided by wikipedia. `https://en.wikipedia.org/wiki/GameMaker:_Studio`. Accessed: 2016-03-29.

[2] Definition of api provided by howstuffworks.com. `http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing1.htm`. Accessed: 2016-03-29.

[3] Definition of ip provided by wikipedia. `https://en.wikipedia.org/wiki/Internet_Protocol`. Accessed: 2016-03-29.

[4] Definition of tcp provided by wikipedia. `https://en.wikipedia.org/wiki/Transmission_Control_Protocol`. Accessed: 2016-03-29.

[5] Definition of udp provided by wikipedia. `https://en.wikipedia.org/wiki/User_Datagram_Protocol`. Accessed: 2016-03-29.

[6] Definition of w3c provided by w3.org. `https://www.w3.org/`. Accessed: 2016-03-29.

[7] Definition of p2p provided by wikipedia. `https://en.wikipedia.org/wiki/Peer-to-peer`. Accessed: 2016-03-29.

[8] Definition of rtt provided by wikipedia. `https://en.wikipedia.org/wiki/Round-trip_delay_time`. Accessed: 2016-03-29.

[9] Definition of rss provided by wikipedia. `https://en.wikipedia.org/wiki/Resident_set_size`. Accessed: 2016-03-29.

[10] Definition of cpu provided by wikipedia. `https://en.wikipedia.org/wiki/Central_processing_unit`. Accessed: 2016-03-29.

[11] Definition of ram provided by wikipedia. `https://en.wikipedia.org/wiki/Random-access_memory`. Accessed: 2016-03-29.

[12] Definition of lan provided by wikipedia. `https://en.wikipedia.org/wiki/Local_area_network`. Accessed: 2016-03-29.

[13] Definition of gui provided by wikipedia. `https://en.wikipedia.org/wiki/Graphical_user_interface`. Accessed: 2016-03-29.

[14] Definition of ide provided by wikipedia. `https://en.wikipedia.org/wiki/Integrated_development_environment`. Accessed: 2016-03-29.

[15] Definition of ietf provided by techtarget.com. `http://searchsoa.techtarget.com/definition/IETF`. Accessed: 2016-03-29.

[16] Definition of dnd provided by wikipedia. `https://en.wikipedia.org/wiki/Drag_and_drop`. Accessed: 2016-03-29.

[17] 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].

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

[19] 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].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[41] 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].

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

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

[44] 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].

[45] 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].

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

[47] 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].

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

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

[50] 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.

[51] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[67] 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.

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

[69] 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.

[70] Node.js: 1 million concurrent connections. `http://blog.caustik.com/2012/08/19/node-js-w1m-concurrent-connections/`. Accessed: 2016-03-19.