NTNU – Trondheim
Norwegian University of
Science and Technology

Impact Of Latency In Wireless Networks For Real-time Multiplayer Games On Mobile Devices.

Erlend Børslid Haugsdal
Martin Nybø Vagstad

Master of Science in Computer Science
Submission date: February 2015
Supervisor: Alf Inge Wang, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Definition

The goal of this project is to discuss problems related to latency in wireless networks for real-time multiplayer games on mobile devices and suggest implementations that address these problems.

The project will test different implementations with regard to user experience. In a perfect implementation the user would not be able to distinguish between a network game and a local game. In a perfect implementation there is no warps, zero delay on action and all users see the same game state.

With current technology, this is not achievable over the Internet. This project will discuss methods of minimizing and disguising inconsistencies in ways that improve the user experience. The game does not need to be perfect, so long as the user perceives is as perfect. The project will mainly look at these challenges from the perspective of a simple real-time racing game.

Abstract

This project describes the current state for wireless networks with regard to real-time multiplayer games on mobile devices. It looks at the limitations of these networks and mechanics to reduce the impact of the introduced network latency. The goal is to find an implementation where the user is not able to distinguish between an offline, perfect version and an online version where the user plays against other users in real time. To find this implementations the authors discuss the choices a developer must make when creating a networked game. Three different network implementations are implemented in a simple racing game created by the authors, and the game is tested on a set of users. The results indicate that users focus on responsiveness before noticing inconsistent game states.

Sammendrag

Dette prosjektet beskriver dagens tilstand for trådløse nettverk med tanke på sanntids flerspillerspill for mobile enheter. Det ser på begrensingene til disse nettverkene og mekanismer for å redusere påvirkningen av forsinkelsen i nettverket. Målet er å finne en implementasjon hvor brukeren ikke klarer å skille mellom en perfekt, lokal versjon og en nettverksversjon hvor brukeren spiller mot andre i sanntid. For å finne denne implementasjonen diskuterer forfatterne valgene som utviklere må ta når man lager nettverksspill. Tre ulike nettverksimplementasjoner er bygget inn i et enkelt kappløpsspill laget av forfatterne, og spillet er testet på en mengde personer. Resultatene fra testen indikerer at brukere fokuserer på respons på egne handlinger før de merker ugyldige spilltilstander.

Preface

This master's thesis was written in the time period from September 2014 to February 2015.

Stakeholders

Erlend B. Haugsdal	Erlend is one of the authors of this report. He is studying Computer Science at NTNU.
Martin N. Vagstad	Martin is one of the authors of this report. He is studying Computer Science at NTNU.
Alf Inge Wang	Alf Inge is the course staff representative and the advisor during the project.

Acknowledgments

We would like to thank our advisor Alf Inge Wang for his support and guidance throughout the project.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Problem Description

This chapter contains the authors' personal motivations for doing this master's thesis, a list of definitions for the project and a reader's guide.

1.1 Motivation

As smartphones get more powerful and new mobile networks are built, the boundaries for what is possible to create are constantly moved. The mobile gaming industry is growing fast and developers can reach billions of users. By allowing users to play together in real time, a whole new dimension is added. Players can compete or cooperate across different genres and interact with each other.

To create a great real-time multiplayer experience one must first understand the limitations of the networks and how this impacts end users. This master's thesis will looking at different mechanisms to mask the network delay and how these mechanisms influence each other. Creating a fair, yet responsive game which the user loves is the goal of any game developer. Our goal is to help developers do this.

Understanding what users have access to and how it is possible to create the best experience while reaching the highest amount of users is the key when creating new real-time multiplayer games. The authors of this project work in a gaming studio and have a personal and professional interest in how to optimize the experience for networked games.

1.2 Definitions

Client-server Communication model where a server provides a service and clients request access to it.

Client An entity which accesses a service. In this project, a client is usually the mobile device of a user.

Dedicated servers The server code runs on separate instances, not on the clients.

P2P Peer to peer, direct communication between two or more clients.

RTT Round-trip time, the time a packet needs to reach a receiver and then return to the sender.

Ping A program that measures RTT. Can be used as a verb.

DTTW Don't Touch The Walls, the game created for the master's thesis. Described in Chapter 6.

ITU International Telecommunication Union, is the United Nations specialized agency for information and communication technologies.

1.3 Reader's Guide

Part 1 - introduction The first part of this project introduces the goal of the project and how it will reach the goal.

Chapter 1 contains the authors' personal motivations for doing this master's thesis, a list of definitions for the project and a reader's guide.

Chapter 2 describes the research questions and methods used in the project.

Part 2 - Prestudy This part looks at research done on real-time multiplayer games for mobile and the technologies available for the users.

Chapter 3 describes the current state of network latency and ways to handle this. It looks at previous research on the area with regard to player tolerance and with a focus on mobile networks.

Chapter 4 will look at the network technology used by mobile devices and how well it is spread throughout the world.

Part 3 - Testing Latency and Game Prototype This parts contains a latency test on WLAN and the most common mobile network, and a suggestion for a real-time multiplayer game to be tested.

Chapter 5 describes the latency test and results.

Chapter 6 describes the implementation and thoughts behind the game Don't Touch The Walls.

Part 4 - Suggested implementations for optimizing user experience This part looks at multiple network implementations for Don't Touch The Walls, the user experiment and evaluation of the project.

Chapter 7 describes different optimizations that can be made when designing a real-time multiplayer game. Each topic is first explored in general and then in relation to Don't Touch The Walls.

Chapter 8 suggests three possible network implementations for the game Don't Touch The Walls.

Chapter 9 describes the experimental part of this project where the implementations suggested in Chapter 8 are tested and compared.

Chapter 10 contains our evaluation of the latency test performed in Chapter 5, the implementations of Don't Touch The Walls described in Chapter 8, and the user experiment performed in Chapter 9.

Part 5 - Summary This part contains our conclusions and suggestions for further work based on our experiences from this project.

Chapter 11 presents our conclusions.

Chapter 12 suggests further work that can be done from the results from this project.

Chapter 2

Research Questions and Methods

This chapter describes the research questions and methods used in the project.

2.1 Research questions

These questions try to narrow down the main sources of impact for the user experience of a real-time multiplayer game. The thought behind each question is explained in detail below.

RQ1 What is the latency of UDP and TCP on mobile devices on different network types? (Modern devices and networks)

RQ2 How do the different latencies impact gameplay and user experience?

RQ3 What is the optimal implementation for real-time multiplayer mobile games?

What are the pros/cons of optimizing for:

- **RQ3.1** Latency
- **RQ3.2** Responsiveness/response time
- **RQ3.3** Network data size/number of updates
- **RQ3.4** Correctness/fairness

RQ4 Which combination of these gives the best user experience?

2.1.1 RQ1: Latency in modern networks with modern devices

This information is critical as different latencies allows for different approaches. Mobile networks are in rapid development as new technology is deployed. As

latency in mobile networks decreases, developers can build better experiences for end users.

2.1.2 RQ2: Impact of latency

With RQ1 answered, the authors can now look at specific latencies and how they will affect the user. By reading others papers and performing user experiments we hope to understand the impact this has on games. This will show what needs to be solved by latency handling mechanics.

2.1.3 RQ3: Optimal implementation

When optimizing one part of an implementation there is often a tradeoff involved. It is important to understand the impact the solutions have on each other, so that the overall implementation is as good as possible.

2.1.4 RQ4: Which combination of these gives the best user experience?

Answering this question is perhaps the main goal of the project. By using the information gained in RQ1-3 and a user experiment we hope to help developers understand what they should think about when creating a game. The question itself does not have a definitive answer as it depends on the game itself. We still hope that we can give some general guidelines for best practice, for both implementation and feel.

2.2 Research Method

We will start by doing a literature search, looking for recent papers detailing work done on latency handling mechanisms and the impact latency has on the user experience. Here we will look at papers for both PC and mobile applications.

When looking for literature, we will use the Google Scholar, IEEE, and Springer search engines. We will also look at the websites of game companies and game engines to understand how they solve these issues.

After mapping the relevant mechanisms, we will do a test where we measure the latency for some of the most common mobile networks. This test is based on the work of Wang et al. [1] and will be compared to a the test they preformed in 2009 to see how the networks have evolved. Based on the this data, we will discuss which of the latency handling mechanisms will give a good experience on mobile networks.

We will create a simple real-time racing game where network inconsistencies are highlighted. The game will have four game modes. One offline against an AI and three implementations of promising latency handling mechanisms. We will then do a user experiment where the test subjects try the offline version and one online version before answering questions comparing the experiences. These test

result will be the base for determining a good way of handling latency in real-time multiplayer games for mobile.

Part II

Prestudy

Chapter 3

State of the Art

This chapter describes the current state of network latency and ways to handle this. It looks at previous research in the area with regard to player tolerance and latency handling mechanisms.

3.1 Network latency

Network latency is the time packets take to travel from one host to another. The lower bound for latency between two hosts is the speed of light multiplied by the distance between them. For hosts at opposite sides of the earth, the theoretical lowest round-trip time (RTT) is about 133 ms. The actual network latency is significantly higher. This is due to the network layout not giving straight lines to/from every host, delay induced by routers and switches, and the fact that signals do not travel at quite the speed of light. Even with fiber optics, the speed is about 30 percent slower than through vacuum. In practice, one can expect RTTs for wired networks in the order of 100-300 milliseconds to most places in the world, and less than 100 milliseconds in the same continent.

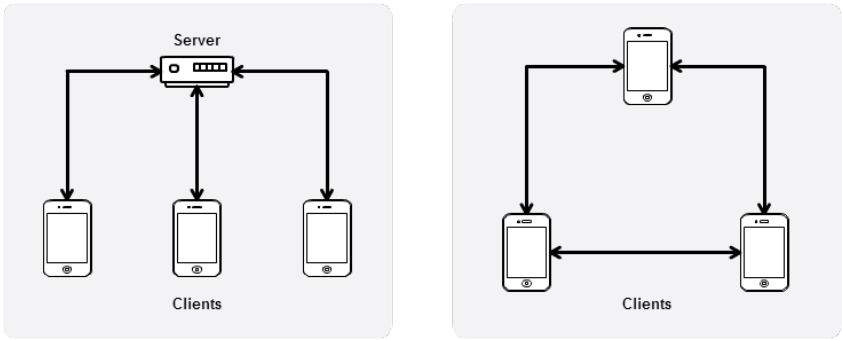

Figure 3.1: Client-server and P2P architecture

Networked games can either use a client-server approach or have players communicate directly, also known as peer to peer (P2P). The two approaches are shown in Figure 3.1. Client-server can be done using dedicated servers or by having one player act as server. Dedicated servers give developers additional control, but increases the latency as all packets must go through an extra node. P2P minimizes latency for all players as all packets go directly to the intended receiver, but increases network traffic as all packets must be broadcast to all players.

No matter the approach, the distance between communicating devices is crucial for low latencies. Games that run dedicated servers usually have servers set up in different parts of the world to ensure that all players have a server nearby where the latency is acceptable. For instance, Valve's game Dota 2 [2] uses 9 different server locations, with 3 in Europe alone, to minimize latency for all players [3]. With servers spread out well, most players are able to find servers with less than 75 ms RTTs. Players also expect this out of modern games. Armitage's experiment [4] on latency sensitivity in the multiplayer game Quake 3 [5] suggests that players actively prefer game servers with less than 150 ms latency. The same numbers are relevant for P2P implementations as peers should be located close to each other geographically in order to reduce latency.

Example Using StartPing [6] to ping Valve's server locations [3], one can see that any country in Europe achieves RTTs of no more than 60 ms to servers in Luxembourg. Similarly, RTTs from the east to west coast of USA are in the order of 70-95 ms. By having server locations on both coasts, one can offer low latencies for most players.

For direct communication (either client-server or P2P), it is crucial to match player on geographic location to ensure that they are all located in the same area. Ideally, this approach can match players in the same cities or in the same room, giving very low latencies. This topic is explored by Manweiler et al. [7] who propose a system for matchmaking in mobile games.

Matching players on location segments the user base and increases the amount of players needed for players to be able find acceptable opponents within a reasonable amount of time. This is something developers need to consider before dividing into too many regions or too many matchmaking options.

Mobile networks do not change any of this, but can add significant latency at the edge of the network. Packets take longer to travel from the device and onto the Internet. Wang et al. [1] measured this overhead in 2009 and concluded that games that use 2G or 3G networks must tolerate latencies of at least 300 ms. Chapter 5 recreates this experiment and measures the overhead for different types of mobile networks in 2015. For mobile networks with very high latencies such as GPRS (latencies of > 1000 ms) [1], the latency caused by distance is less significant.

3.2 Player tolerance

Players experience latency in two main ways: Delay on own actions and delay on actions of others including the system. Delay on your own actions means that the system waits before applying your commands in order to allow the command to reach the other players so that the command is executed at the same time on all systems, increasing consistency. Pantel et al. [8] state that input delays of over 100 ms should be avoided for real-time racing games, as this compromises the game experience for players. At 250 ms, expert players stop playing, claiming that the controls are not reasonable anymore.

Delay on actions of others can result in inconsistent views where positions of opponents are no longer where you see them on your screen. In first person shooter (FPS) games, this can result in cases where you shoot directly at someone, but miss because they are no longer there. The severity of inconsistencies depends greatly on the implementation and the amount of latency. Wang et al. [1] describe that for most wired online games, gameplay gets affected at latencies over 150 ms. They also conclude that games that are to run on UMTS networks must tolerate at least twice this latency (> 300 ms). This implies that networked games that run on UMTS need to think of new ways to handle latency in order to provide a good player experience.

It is worth noting that player tolerance varies between different game types. For FPS and racing games, players are very sensitive to latency, where real-time strategy (RTS) games can tolerate more. Claypool & Claypool [9] state that different actions have different latency requirements. They categorize actions by the precision of the action and the deadline for completing the action. FPS games generally have high precision and short deadlines for resolving actions, while RTS games have longer deadlines. They state that gameplay quality becomes unacceptable for FPS games at latencies over 100 ms, while RTS games can tolerate 1000 ms. Sheldon et al. [10] examine the effect of latency on players of the RTS game Warcraft III [11] and state that while delays of several seconds are noticeable for players, it has little to no effect on the outcome of games. The overall strategy seems to be more important than quick reactions.

3.3 Latency handling mechanisms

There are many ways of handling latency in games, but the bottom line is that you need to visualize the actions of other players. Receiving a packet of information about a player's state or action is called an update. In between updates, the local player does not know what the other players are doing and a natural way of handling this is extrapolating.

3.3.1 Extrapolation

Extrapolation is an attempt to estimate a future game state. Since all clients are running the same code, they follow the same rules. Given a player's state and position, one can estimate the new states of the player given no new player input.

A common approach to extrapolation is dead reckoning [12]. This approach extrapolates positions of all game objects, and sends updates if input changes the state or course of an object by more than a set amount. In practice this may involve updating for every player action, but it can also tolerate small movements without requiring updates.

The downside to extrapolation is that when a command does come through, the object in question has already been extrapolated assuming no input. If the change of state or course is large, it will cause player or object to move significantly.

The easy way to fix inconsistent states is to simply set the player's state to the new state received in the update (also called warping). This is very visible to the user and can be experienced in a negative way. If the difference is small, one can usually interpolate between the two states in order to hide the warp from the players.

Pantel et al. [13] explore dead reckoning schemes and prediction methods that increase consistency for different game types. Yahn Bernier explains the latency handling mechanisms used in the Half-Life game engine [14]. The engine relies on dead reckoning to keep players looking lively in between server updates. By running the same movement code on the client and the server, clients can perform actions locally at the same time as they send action requests to the server and assume that the server performs the actions in the same way.

This engine also compensates players for latency by rolling back the server and allowing players to act on objects as they were at a previous point in time. The latency compensation window is interesting as it also opens up the possibility of cheating. A client can receive an update containing the absolute position of an opponent at a certain time, and then send a message shooting the opponent timestamped at the time the opponent was there.

3.3.2 Lockstep

Latency can also be handled without guessing what players are doing. One solution to this is the lockstep method [15]. All clients wait for updates from all other clients before performing the next set of operations. That is, we lock the system, wait for all clients to send their input, and then perform a step. Using this approach, there is no need to send data about all objects in the game since it guarantees that all systems have the same state when executing operations. It is sufficient to send the commands that the players are executing.

The method is very effective at keeping the state correct, but suffers enormously in environments with high latency, as all clients must wait for the slowest client for every update. If there is packet loss, this can mean spikes of several seconds for all players. If a game requires updates 60 times per second, that implies that the latency for the slowest client can not exceed 16,6 ms. The approach is traditionally

used for LAN settings, as this latency is unrealistic for any Internet setting.

The latency limit can be improved by reducing the number of updates per second, or more importantly by using input delay (more on this in Section 3.3.3) to send commands that will be executed at some point in the future. This approach is used by many modern RTS games.

3.3.3 Input delay

Mauve et al. [16] describe a technique called local-lag which explores the tradeoff between responsiveness and the appearance short-term inconsistencies in games. If no local-lag is applied, an action done by one client is visible to another client after a delay of at least 1/2 RTT.

By delaying the execution of a command locally while sending the update immediately, both clients experience the same delay, thus they see the same game state. This is done by specifying when an action is supposed to be executed. Player A sends a message saying that he will jump at time T = current time + local-lag delay. Player B receives the message before time T, and schedules the action to take place at time T. This solution requires synchronized clocks. If Player B receives the message after time T, action needs to be taken in order to recover the game state. Mauve et al. [16] suggest a timewarp algorithm to fix this problem by rolling back to a valid state and reapplying the commands at the correct times.

An important part of local-lag is determining a good value for the applied input latency. The delay needs to be large enough to allow clients to receive updates before they should execute. This implies that it needs to be larger than 1/2 RTT or all updates will be received too late. It should also be as low as possible in order to keep the application responsive. As described in Section 3.2, input delays of over 100 ms should be avoided for real-time games. Mauve et al. [16] agree that latencies of 80 - 100 ms will not be noticeable for the user, and the number may be higher for certain applications.

Khan et al. [17] propose a method for dynamically adjusting the input delay for local-lag to account for changing network conditions. Different types of objects can also use different local-lag delays as some objects are more important than others.

3.3.4 Interpolation

Interpolation is often used to reduce the number of network updates and avoid warping. Instead of sending updates every frame one can send fewer updates and interpolate between the current state and the new state. This is done by creating a set of new states between two known states, and moving the object through them over time. As long as the path is not blocked this will seem more natural then instantly moving the player to the new state.

Chapter 4

Technology

This chapter will look at the network technology used by mobile devices and how it is spread though out the world. This is relevant as it is important to understand what the users have access to and how it will affect the user experience.

4.1 Mobile networks and coverage

Mobile networks allow users to communicate over great distances. The technology has been in development since the 1980s, and is about to reach its fourth generation [18]. Each generation symbolizes a large leap and there are multiple networks per generation. Today's mobile devices often support multiple generations and networks as the coverage can vary based on location. This sections will give a short summary highlighting the most used networks.

2. Generation 2G cellular telecom networks were commercially launched on the GSM (Global System for Mobile Communications) standard in Europe and CDMA (Code Division Multiple Access) in the USA [19]. The main features were that the phone conversations were digitally encrypted. It was significantly more efficient on the frequency spectrum allowing for far greater mobile phone penetration levels. It also introduced data services for mobile. This made it possible to do more than just call over the network. The SMS (short message system) and later MMS (multimedia messaging service) were added. Users could now send messages to each other. An SMS only supports 160 characters and as such the 2G networks were built mainly for voice services and slow data transmission. Examples of 2G networks are GPRS (General Packet Radio System) and EDGE (Enhanced Data for GSM Evolution).

3. Generation With the introduction of 3G networks, data transmission got a huge improvement. To be classified as a 3G wireless network, the network has to meet the IMT-2000 standards, with a minimum speed of 2Mbit/s for stationary or

walking users, and 348 kbit/s in a moving vehicle [20]. UTMS (Universal Mobile Telecommunication System) is the standard 3G network and builds on different technologies, such as HSPA (High Speed Packet Access) and HSPA+ for fast packet rates.

4. Generation The 4G is currently under deployment and offers a higher peak data rates then the 3G networks. ITU (International Telecommunication Union) have suggested a new standard, the IMT-Advanced, with a minimum speed of 1 Gbit/s for stationary or walking users, and 100 Mbit/s in a moving vehicle [21]. Example of a 4G network is LTE-Advanced.

A comparison of the latency of these networks can be found in Chapter 5.

Coverage Reports from the ITU, shows that by the end of 2014 there are almost 7 million mobile-celluar subscriptions worldwide [22]. Figure 4.1 taken from this report shows the mobile-celluar subscriptions, total and per 100 inhabitants for 2005-2014, where 2014 numbers are an estimate.

Figure 4.2 taken from Ericsson network traffic [23] shows how these users are spread across the different networks. Most of the users only have access to GSM/EDGE, which might give poor performance in a real-time multilayer game. Less then half of the users have access to 3G or better networks.

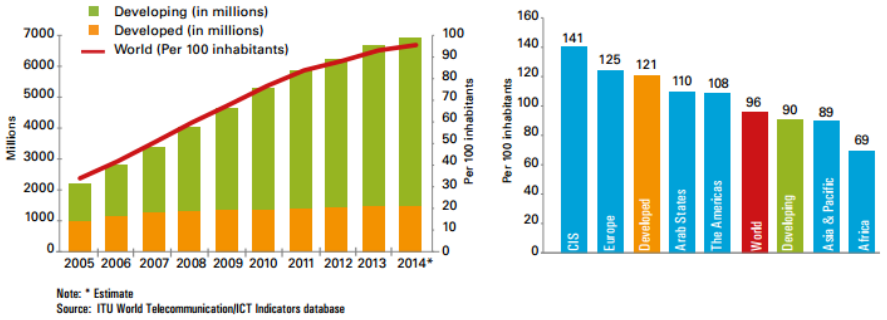

Figure 4.1: Users connected to mobile networks

4.1.1 WLAN

WLAN or wireless local area network is supported by most modern smartphones, and does generally not charge users for traffic. Compared to mobile networks that have range in kilometers, WLAN offers range in tens of meters. WLAN usually follows a separate subscription where the user pays for a certain bandwidth. WLAN usually offers significantly lower latency than mobile networks, but support fewer simultaneous users per access point.

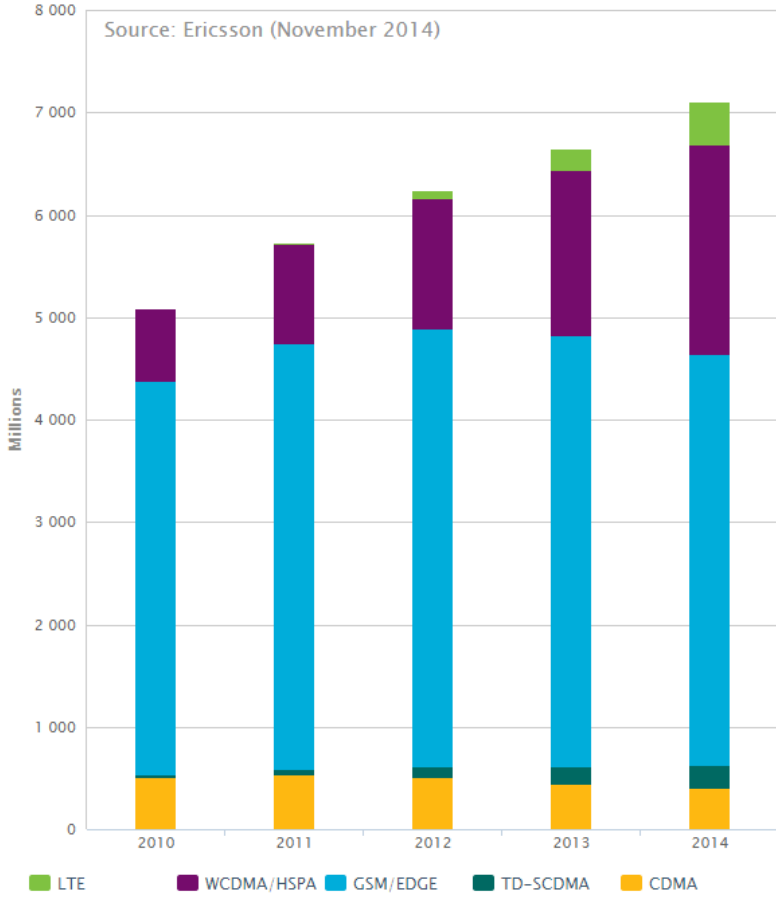

Figure 4.2: Distribution of users on different mobile networks

4.2 Mobile devices

This project looks at games for the current era of smartphones, namely iOS and Android devices. This era started when the first iPhone was released in 2007 [24]. These devices featured touch screens, high resolution, and no physical keyboards. Apple also introduced their own store where developers could easily distribute their own programs, such as games. The first iPhone supported WLAN and GSM/GPRS and EDGE. In 2008 Apple released iPhone 3G which supported the next generation mobile network such as UTMS/HSPA.

The phone was now powerful and had good enough network capabilities to support real-time multiplayer games. Since then, new devices have continued to support new and more powerful networks. Other actors such as Samsung, HTC, Nokia and LG have also released phones with different operating systems, but the

same capability for gaming in both performance and networking.

4.2.1 Summary

Most modern smartphones have the power and network capabilities to run simple real-time multiplayer games. The limiting factor for many user is instead the coverage of 3G or better network. Many users live in areas where they can only use 2G networks and this might reduce the experience as they only support a low data throughput.

Part III

Testing Latency and Game Prototype

Chapter 5

Measuring latency

This chapter is a modern reconstruction of the experiment performed by Wang et al. [1] With more powerful hardware and more modern network infrastructure it is reasonable to assume that the results would be different from those measured in 2009.

5.1 Test setup

Figure 5.1 shows an overview of the test setup. The rest of this section describes the setup in detail.

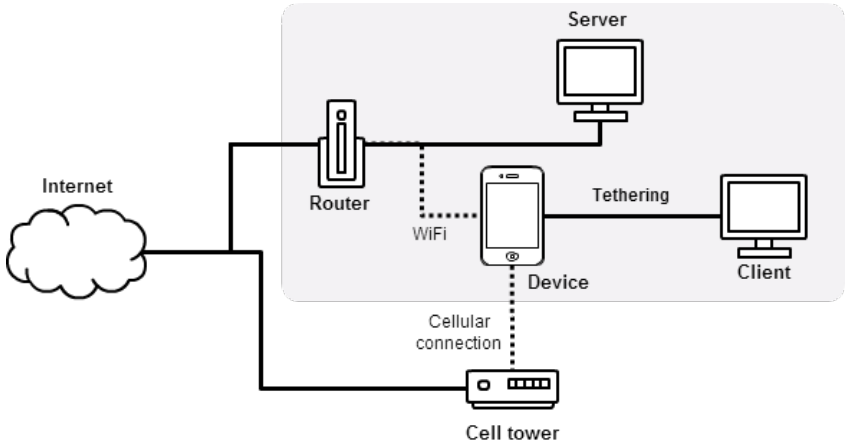

Figure 5.1: Test setup

5.1.1 Hardware

Two different phones were used as devices for the test, one LG Nexus 4 and one Samsung Galaxy S5, both running Android 4.4. Multiple phones were used to get access to more cellular networks. The phones were connected to a computer using USB tethering. The phones shared their network connection with the computer and the testing software was running on the computer.

Initial tests showed that using tethering as opposed to running the tests directly on the phones added zero to one millisecond of delay on a round-trip. Since the overhead was so small and because tethering greatly simplified the testing procedure, this was the chosen approach.

5.1.2 Network

Android phones have the choice of force 2G or using the best available network. The test are therefore run on the networks that are available on the location of testing. The tests were performed in Bergen, Norway using both Telenor and Netcom's mobile networks. The available networks were EDGE, HSPA+, LTE and WLAN. Based on the coverage found in Chapter 4 this is sufficient to discuss the state of latency in modern networks.

The setup used here is aimed at minimizing latency related to distance between hosts, instead focusing on latency related to the mobile device and the technology it connects to. When testing the mobile networks, packets had to travel to a nearby cell tower and through the Internet back to the router and finally to the server over WLAN. When testing WLAN, the packets only travel through the router, keeping the distance at an absolute minimum.

5.1.3 Software

A simple client and server program written in Java was used to perform the tests. The client sends packets at fixed intervals, while listening for incoming packets. The server simply responds to any incoming packet by sending the same packet back.

The client sends packets without checking if the previous packet has made a round-trip, meaning that packet loss for TCP leads to delays for any packet sent after the one that got lost. These packets are stored by the receiver, awaiting retransmission of the missing packets since TCP guarantees that packets arrive and that they arrive in order.

Each packet contains a 4 byte integer representing the packet number starting at 0 and incrementing. This is used to keep track of RTT for UDP and is kept for TCP in order to keep the payload size the same. For TCP, the no-delay flag was used for both the client and the server application.

The message sending interval was varied to test its impact on RTTs. This also makes the results easier to compare to those of Wang et al. [1] in order to look at the progress mobile networking has made over the past five years.

5.1.4 Test parameters

Transport protocols TCP, UDP

Networks WLAN, LTE, HSPA+, EDGE

Intervals 15 ms, 50 ms, 100 ms, 150 ms, 200 ms, 250 ms, 300 ms

Iterations 100-1000

5.2 Results

This section presents the results from the test. They are graphed in Figure 5.2 and summarized in Table 5.1.

The test shows that WLAN has the lowest latencies by far. LTE and HSPA+ networks are very close to each other in terms of latency, while EDGE is slower by a large margin. The difference between TCP and UDP is not as significant as measured by Wang et al. [1], but UDP still performs better in all the tests. Most packets have very similar RTTs with UDP and TCP, but TCP suffers more in the event of packet loss and this impacts the average times.

The sending interval seems to have limited impact on the test results. The obvious exception is the 15 ms interval for EDGE networks. This interval pushes the bandwidth close to the limit when using UDP and over the limit when using TCP. This is due to the extensive overhead introduced by the TCP protocol and the limited bandwidth of the EDGE networks.

Compared to the numbers from 2009 [1], these results suggest a huge improvement in overall latency. This can be due to improved network infrastructure and protocols as well as more powerful mobile phones. WLAN numbers are significantly improved, and it is still the superior network by far. HSPA+ networks have taken over for pure UTMS, and LTE networks are the next generation of networks which are still being deployed many places. LTE networks have similar latencies as HSPA+, but feature much higher bandwidth.

Protocol	TCP				UDP			
Network	WLAN	EDGE	HSPA+	LTE	WLAN	EDGE	HSPA+	LTE
Min	0	197	94	65	0	134	79	62
Max	157	1095	123	103	121	942	166	112
Average	15	246	90	89	9	167	87	81

Table 5.1: Round-trip time at 100 ms interval

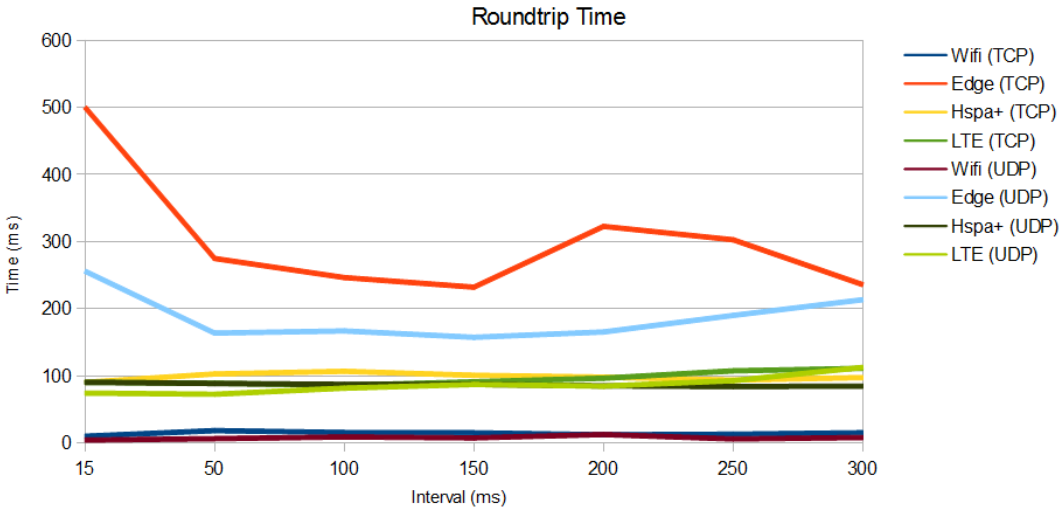

Figure 5.2: Measured RTT

5.2.1 Impact

As seen in Table 5.1, LTE and HSPA+ networks have average round-trip times of less than 100 ms and relatively low jitter. This combination is attractive for real-time multiplayer games. As discussed in Section 3.2, this is a tolerable latency and allows use of techniques like local-lag.

EDGE networks come close when using UDP, but feature a high amount of jitter. The jitter is less important for UDP as long as packets are non critical and can be tossed away if they arrive late. When using TCP with EDGE networks, the jitter can cause spikes of over a second which is very noticeable for users.

One should keep in mind that these numbers do not include distance-based latencies between players and servers. This latency needs to be added on top when designing a networked game. As discussed in Section 3.1, distance-based latency can be limited to 50 - 100 ms by matching players on geographic location. When this is added to the RTT measured in the test, HSPA+ and LTE networks have RTTs of less than 200 ms. Using a peer to peer approach, this means that packets arrive at other hosts in 1/2 RTT which is less than 100 ms. This allows for use of techniques such as local-lag with input latencies that are acceptable according to Section 3.2.

Chapter 6

Choice of real-time multiplayer game for testing

6.1 Genre

When choosing a game to test for this project, the two most important factors are that the game is able to highlight network issues and challenges, and that it is relatively easy to learn. The authors have experience building racing games so this is a natural area to explore. Racing games feature:

Moving objects Once things are moving, latency or jitter can cause large inconsistencies. Higher speeds mean more impact of latency.

Collisions Objects can collide with walls or each other. Synchronizing collisions can cause lots of interesting issues.

Simple controls This is very beneficial for a mobile game. Test subjects will be able to look past the control scheme and experience the game.

Ability to see other players a lot Latency based inconsistencies are mainly a problem when players can see them. Racing games lets the players see opponents and their positions a lot of the time.

Synchronized start Network issues become abundantly clear when a race starts as one would expect all players to start at the same time.

In order to keep the game and the implementation simple, limiting the game to 2D is a natural choice. Using a top-down camera in a fixed position which shows the entire playfield allows players maximum visibility of others, letting them detect network issues easily.

6.2 Don't Touch The Walls (DTTW)

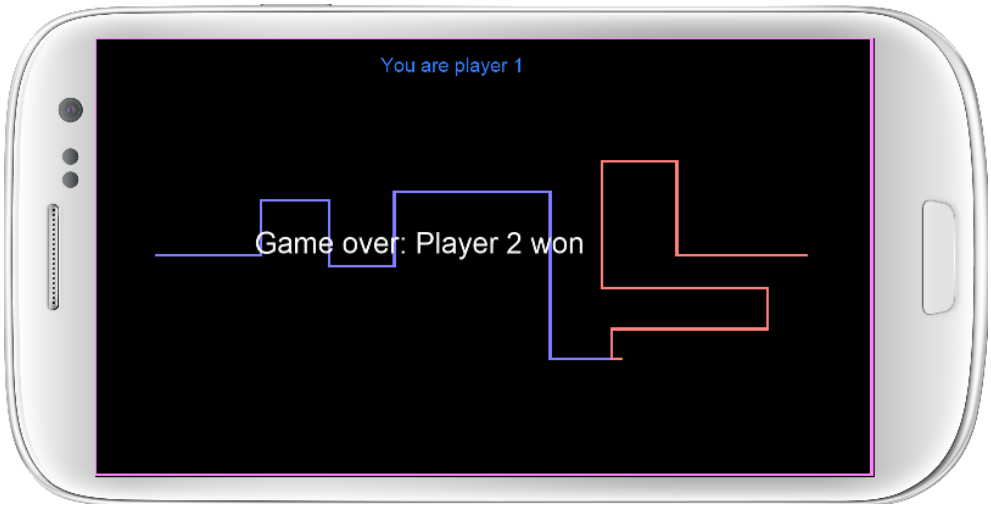

Figure 6.1: Don't Touch The Walls screenshot

6.2.1 Rules

The game is closely based on Armagetron [25] which is a game that emulates and expands on the light cycle scenes from the movie Tron [26]. Players control a block in a 2D playfield. The playfield is a matrix of 240 x 160 tiles. Each tile can contain a wall, a player, or nothing.

The game runs 60 frames per second. Players move forward automatically with a constant speed of 1 tile per frame, and can turn 90 degrees in either direction. While moving, players leave a permanent trail behind them which is lethal to anyone who collides with it. Colliding with the walls of the playfield is also lethal. The goal is to be the last player standing.

6.2.2 Considerations

The entire playfield is visible for both players at all times, and the top-down camera does not move.

The players move relatively quickly, and this is a game of reactions. As the game progresses it becomes increasingly difficult as there are more player-made walls to avoid. Players need to plan ahead and try to anticipate the moves of the opponents while not blocking themselves in.

The game can be played with any number of players, although there should be at least two. For the purpose of this project, the number of players is limited to two. A game with two people will be able to highlight all the same network issues and challenges as a game with more than two. By limiting to two players

complexity is reduced and test subjects have fewer things to worry about, which might help them to notice network related inconsistencies.

Significant network latency will impact the game, and different implementations will give different user experiences. If the two players do not share a common game state, players can make moves that seem impossible to the opponent. For example, player A can run into an area that player B believes to contain a wall. If players have a large input delay, making quick decisions on where to move becomes difficult.

Warping will also have a large impact, as a player might see the opponent quickly moving a large distance leaving a wall in an area thought to be empty. Depending on the implementation, this can also lead to walls being removed from areas where they were placed.

6.2.3 Implementation

Technology Corona SDK [27] was used to create DTTW, as both the authors have experience with this tool. Corona SDK is an engine for 2D games. It uses Lua as a scripting language and compiles to both Android and iOS. The server application was written in Java and ran on a local computer.

Client implementation When the user selects a game mode, the client connects to the server and is assigned a player id. This can be either 1 or 2. The client creates a player object for each player that stores the position, current direction and a list for all the turns. When two players have connected, the server send a start game message that starts the game loop. The game runs at 60 fps, which means that the game loop is run every 16,67 ms.

The game loop first checks for queued turns for each player. If a turn is found that should have been executed on a previous frame, the client rewinds and executes the turn at the correct frame. If the turn is at the current frame, the new direction is stored in the player object. It then places a wall on the players position before it finds the next position based on the direction. The new position is checked against the map state to find out if the move is valid. If there is a wall there, the game is over. Otherwise the player is moved to the new position. Depending on the game mode this collision check can be done by the client or the server.

When a player clicks to turn, the frame for the turn is stored in the queue and sent to the server/opponent. This part of the implementation can vary based on the game mode.

Server implementation The server implementation has two different modes. For game implementations not requiring any server interaction it serves as a packet switch and a way of easily connecting to opponents. Two and two players are matched together and incoming packets are simply forwarded to the opponent. This setup closely emulates a pure P2P environment while greatly simplifying connecting and finding other players.

The other mode runs the game logic on the server. The same game loop used on the client is used on the server. Depending on the selected implementation,

packets are broadcast when they are received or when their actions are performed.

Both modes have the ability to add latency to incoming or outgoing packets. This can be done by specifying a mean amount and a standard deviation if jitter is required.

Part IV

Suggested implementations for optimizing user experience

Chapter 7

Tradeoffs of optimization approaches

This chapter describes different optimizations that can be made when designing a real-time multiplayer game. Many of these have positive and negative effects, and this chapter discusses the tradeoffs that developers must consider. Each topic is first explored in general and then in relation to Don't Touch The Walls (DTTW).

7.1 Optimizations regarding network latency

Network latency can be looked at as two separate parts: The time related to the technology used to connect to the network and the time related to the distance between endpoints. Looking at Figure 7.1, the first part is the time it takes to get packets from the device to the router or cell tower. The second part is the time from the router or cell tower to the remote server or peer. Most of the optimizations discussed in this section relate to reducing one of these parts.

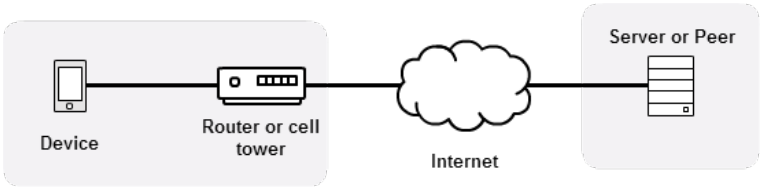

Figure 7.1: Latency components

Improving latency by reducing the distance between endpoints will impact who can play together and segment the user base. Improving latency by restricting the technology used to connect will restrict devices and areas with poor network conditions, thereby reducing the potential user base. The upside is obviously that if developers are guaranteed that all players have solid connections with low latencies,

this can impact the types of synchronization schemes used. It is also likely to improve the user experience for the players that are able to play.

Use UDP instead of TCP According to our tests in Chapter 5, UDP has lower average latencies than TCP. It features less overhead for each packet and sends no acknowledgment packets. There is also no connection handshaking which makes roaming and reconnecting much simpler than with TCP. The downside is that UDP is not reliable, so if a game requires packets to reach the receivers this needs to be implemented on top.

Limit connection to WLAN or HSPA+/LTE By limiting the type of connections players can use, one can limit the time for packets to get to the network significantly. The tests in Chapter 5 suggest that limiting to HSPA+/LTE can give < 100 ms RTT before accounting for distance. EDGE and other older technologies present much higher latencies and also more jitter. The downside to this type of optimization is that potential players without access to modern networks are unable to play. It may also limit players from playing while on the go if a good connection is not available in the current area. Limiting to WLAN would give PC/console-like latency conditions, but is very restrictive for players. For games aimed at being played on a tablet in the home where there is WLAN, this might seem less extreme.

Dedicated servers in different geographical regions Games using dedicated servers will improve latencies for all players by supplying servers in different geographical regions. One location per continent should be the minimum. This ensures that most players worldwide are able to find a server with less than 100 ms in distance related latency. There are two downsides. The first is that it divides the player base for matchmaking and such. The second is that more locations increases complexity and maintenance, which usually increases costs.

Peer to peer with geographical matchmaking For peer to peer implementations it is possible and quite beneficial to match players on geographic location. By having players measure round-trip times to potential opponents one can guarantee that no two players have a RTT of above some desirable value. The downside is segmentation of the user base and longer queue times. The tradeoff here is how low one wants to force the latency to be as segmentation is directly tied to latency bounds.

Limiting multiplayer to WLAN or Bluetooth By requiring all clients to connect via Bluetooth or to be connected to the same WLAN, latency related to distance is reduced to almost zero. While this allows for great synchronization, this optimization has the obvious downside of requiring players to be physically close to each other. This can also be thought of as an extreme version of P2P with geographical matchmaking.

Don't Touch The Walls (DTTW) This is a type of game where the actions of the opponent immediately affect the player and can change what the optimal next move is. It would be constructive to try to guarantee a maximum latency to ensure a good user experience. Segmenting the users on geographic location to reduce distance seems worthwhile. Limiting user connections to WLAN or HSPA+/LTE depends on if the developer wants the game to be highly consistent or reach enormous amounts of users. Both are reasonable choices. Limiting to WLAN/Bluetooth seems unnecessary. Using UDP over TCP would lower latency slightly, but might require some implementation to guarantee consistent game states. Since players leave a permanent and lethal trail, it is important that every wall is in the right place.

7.2 Optimizations regarding responsiveness

Mauve et al. [16] state that the optimization of the response time and the avoidance of short-term inconsistencies are conflicting goals. This implies that there are good reasons to reduce responsiveness, even though this immediately seems like the wrong thing to do. This section presents some possible tradeoffs between consistency and responsiveness.

No delay By applying no input delay on actions and executing locally as soon as they happen all actions are delayed by at least 1/2 RTT on other clients. Players notify other clients as soon possible. The benefit of this approach is that the client is really responsive. Some types of games can hide most inconsistencies that appear from this approach by predicting future states. An example is simple racing games. The downside is that there is no global game state and clients see their own version of things. It is hard to say who is right when objects interact. This makes it hard to separate cheating and latency related issues.

Local-lag This approach is based on local-lag [16] which is described in detail in Section 3.3.3. Actions are delayed by a set amount to reduce the number of inconsistencies. As long as the local-lag amount is less than 100 ms players are less likely to notice it. If the local-lag amount is greater than the time from client to client, the game state is consistent. This approach needs a plan in place for resolving packets that arrive after the set execution time. If the game engine allows it, the simplest is to roll back to the time of execution, apply the action and fast-forward back to the current time. Local-lag performs well in low latency environments without being very noticeable. In high latency environments, it can still function as a version of no delay where states are less out of sync.

Lockstep with input delay Lockstep is described in detail in Section 3.3.2. By delaying execution, lockstep can function in higher latency environments, at the cost of responsiveness. Execution needs to be delayed by the RTT of the player with the highest latency. The main upside of this approach is that it guarantees a

consistent game state. This has a number of other advantages. Only actions need to be sent, not the entire game state as all clients have the same state on every frame. There is no warping or rollbacks either. The main downside is that the slowest client can slow down the game for all players. In cases of packet loss or significant jitter, the game can freeze until packets are retransmitted, which can be very noticeable. Input delays can be high enough that the user experience suffers. As discussed in Section 3.2 input delays of over 100 ms should be avoided. With this implementation, this is in the hands of the slowest host.

Authoritative server A server holds a global game state. The server broadcasts the game state or changes to the game state. Players send actions to the server and await a new game state. If packets from the server are delayed, the client can either extrapolate with the option of rolling back, or freeze the game until a new state arrives.

The benefit from this approach are that the server has a global game state and the server is always right. This makes cheating much harder and clients will never see inconsistent states (assuming no extrapolation). In addition, packet loss or significant jitter only affects the player with the network problems. The main challenges are that players can experience variable input delay since actions are executed on the server, and that one needs to supply servers.

Don't Touch The Walls (DTTW) This is a game where warps and rollbacks are very visible and potentially game breaking. When players are warped or rolled back, the walls need to be moved, literally changing the playfield. Players will likely be very frustrated if a wall suddenly warps in front of them.

A no delay solution features great response time. The player can always turn exactly when requested. Since opponents are actually ahead of what the client displays, no delay will lead to situations where the players cross paths without anyone dying. Warps will happen only when there is significant jitter. In the remaining cases, the opponent will be moving smoothly in a delayed state.

Local-lag features low and predictable response times. As long as the latency is lower than the local-lag amount, the game state is the same with both players. Jitter or packet loss will create very awkward warps where walls move.

Lockstep will give bad responsiveness if latency is high. Game freezes will likely be hard to deal with for players as they have no way of knowing when it will start back up.

Authoritative server will also give bad responsiveness if latency is high. Players that experience freezes will not be able to execute actions after the freeze is over until they have the correct game state again. This will likely lead to the player hitting a wall and losing since the game moves quite fast.

7.3 Optimizations regarding data transfer size

Data transfer size is more relevant for mobile devices than for computers or consoles since many users have data caps or pay per byte. Chen et al. [28] show that many

developers do not focus on this, increasing the cost for end users. Bandwidth is also more of a factor and can vary based on what network the user is connected to. This is discussed more in Chapter 4. If dedicated servers are used there is often a cost attached to sending or receiving data, so optimizing data transfer size can save money for both parties.

Encoding Packet size can depend greatly on how the data is encoded. When sending data there needs to be a common understanding of how to read incoming data. Using existing encodings such as XML or JSON can simplify the process for developers at the cost of increased packet size. It is possible to create custom encodings or formats for sending data where only the necessary bytes are sent, but this will often incur extra development costs and less flexibility if requirements change.

Update frequency A simple way to reduce bandwidth is to reduce the frequency of updates. The simplest approach is to only send player actions. This limits the update frequency to how quickly players can issue commands multiplied by the amount of players. These updates can also be buffered to reduce the number of actual packets sent. This approach assumes that all players have a shared game state so that the actions are applied in the same way. The alternative would be inconsistencies.

An alternative approach is to send the game state regularly. This scales poorly if the game state contains many objects that need to be transmitted with each update. It is possible to use fewer network frames than game loop frames. A game can for example run 60 frames per second while using 20 network frames per second. This approach is more suited to unreliable transfers since the game state will be correct once a packet is received on time.

P2P vs client-server In a peer to peer approach all packets have to be broadcast. This means that outbound data is multiplied by the number of other players when compared to a client-server solution. A client-server solution can also buffer and combine packets to reduce overhead. This approach can also be improved if the server can filter out messages that a client does not need. An example is a player moving in an area where you can not see such as units moving under fog of war in a real-time strategy game.

TCP vs UDP Using UDP reduces overhead for each packet, and avoids sending acknowledgments for received packets. Depending on how developers handle lost UDP packets this can significantly reduce the number of packets sent.

Don't Touch The Walls (DTTW) The game is simple enough that sending the entire game state frequently is possible. One can for example describe the game state as a list of actions and a current position for each player.

It is also possible to only send player actions, which should result in quite few updates in total.

Since the game is limited to two players and the game is so simple, bandwidth and data transfer size is less of an issue than in more complex games.

7.4 Optimizations regarding correctness and fairness

As described in Section 7.2, optimizing responsiveness and correctness are conflicting goals. The impact of inconsistencies can vary. Not every situation will create unfair or unplayable conditions.

For a racing game, the most critical areas are the start of a race and the end of a race. When starting, all players expect to start at the same time and that latency is not clearly visible at this stage. When finishing a race, latency can cause players to see that they finish before their opponents, while in reality, they finished behind them. This type of experience is very damaging because it feels unfair.

If a car has an incorrect position some time during the race, this is far less damaging for the experience. It is important to identify the most vulnerable situations in the game and design the network logic around this. Try to guarantee correctness in the situations that would feel unfair to players.

Who decides what is correct? In peer to peer environments, there is no official server that can decide who wins or loses. This can lead to situations where multiple players think they won or all players think they lost. The approach is also vulnerable to hacking and cheating. All clients need to broadcast all packets, allowing for cheats such as map hacks which are hard to combat.

In peer to peer environments where one peer acts as a server, it is possible to avoid map hacks, but the solution still vulnerable to cheats from the peer acting as server. The hosting player also gets an advantage because his packets always arrive at the server on time, possibly decreasing his response times.

Using a dedicated server, there is one trusted authority that decides who wins and can check incoming packets for cheating. No player is given an unfair position of power.

Don't Touch The Walls (DTTW) For this game, it seems important for all players to have a correct game state at all times. If one player is missing some walls in his playfield, this gives him an unfair advantage. If the client is authorized to report deaths, it is possible to hack the client so that it no longer dies in contact with walls. This kind of cheating would be extremely frustrating to play against, and a dedicated server approach seems reasonable to counter this. At the same time, dedicated servers would increase latency for clients. This would make it harder to synchronize game states without increasing input delay above comfortable levels (See Section 3.2). There are reasonable arguments for both approaches, and it makes sense to test both.

7.5 Summary

Different games have different requirements. In the end, the most important thing is how the user perceives the game. The game does not need to be perfect, but the user should feel that it is fair and responsive.

It is important to find and understand the vulnerable points in the game where players feel the impact of networking decisions. These should be the main drivers for deciding the network implementation.

Chapter 8

Network implementations for the game

This chapter suggests three possible network implementations for the game Don't Touch The Walls based on the game as described in Chapter 6 and the optimizations described in Chapter 7.

8.1 Implementation 1: No delay

This implementation optimizes responsiveness for the local player by not adding any input delay. When the player clicks the screen, the new direction is applied on the next frame and sent to the opponent. Actions from opponents are applied as soon as they are received, and as such they are delayed by 1/2 RTT. If an action is received too early or too late, the game will set the opponent's position and make sure that his walls are correct. That means that if the packet arrives too early, some walls are added to the gap that appears. If a packet arrives too late, the extrapolated walls are removed. Given no jitter, this implementation will not cause any warping because all actions are delayed by the same amount.

Because all opponent actions are delayed you will see the opponent in a position which he has already moved away from. Figure 8.1 shows the game state seen by player 1, with player 2's actual position shown by the dotted line.

This solution can create situations where the local player thinks he has cut the opponent off, but the opponent has already passed through the wall before it was placed. These situations are more frequent with higher latencies as the remote position is more delayed versus the local position.

Figure 8.2 shows an inconsistent game state where player 1 in Figure 8.1 turned left. He crosses in front of player two locally, but player 2 believes he was in front so both players got out of the situation alive. The resulting game state is obviously invalid.

The players report their own death, allowing the game to continue even if the game state look like the game is over. This approach guarantees that a player can

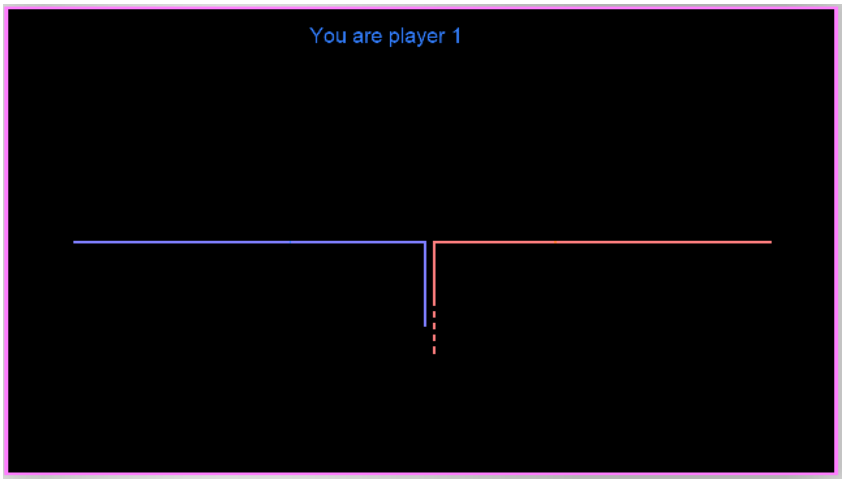

Figure 8.1: Game state for no delay

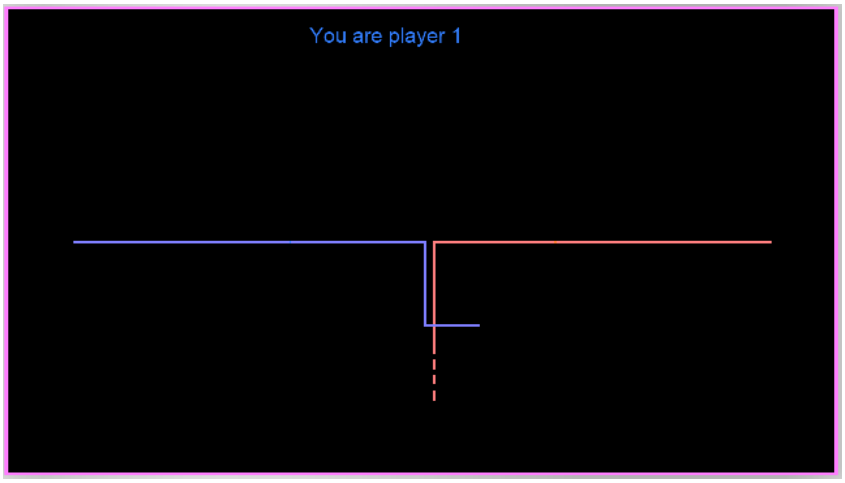

Figure 8.2: Game state for no delay after player 1 has turned

see a wall before running into it.

In low latency situations, this approach will look very good. It suffers in high jitter situations. Jitter will make it seem like the opponent changes speeds during the match.

Since players report their own death and there is no correct game state, there is no need to use an authoritative server and the game should be implemented as P2P to minimize latency.

8.2 Implementation 2: Local-lag

This implementation introduces input delay to reduce inconsistencies caused by network latency. Local-lag is discussed more in both Section 3.3.3 and Section 7.2. Based on player tolerance as discussed in Section 3.2, the local-lag amount should be no higher than 100 ms. This amount can be changed dynamically based on network conditions, but players are likely to get more upset by varying response times. For this implementation the local-lag amount is set to 100 ms. Local-lag performs the best if the local-lag amount is larger than 1/2 RTT.

The clients clocks need to be synchronized at the start of the game so that the game starts at the same time for both clients. By doing this the client will have a common understanding of time for when to execute actions. When the player clicks the screen, the client calculates the position and frame that the new direction will be applied based on the local-lag. This data is then sent to the opponent. Messages that arrive before execution time are queued and executed at the intended point in time. If messages arrive too late the opponent is rolled back to the time of the action, the action is applied, and the opponent is fast forwarded to the current time. This means that the game state is correct in all situations except for cases where there is an action that should have been executed but has not yet arrived.

Figure 8.3 shows the game state as seen by player 1, while Figure 8.4 shows the new state after a turn message arrived too late.

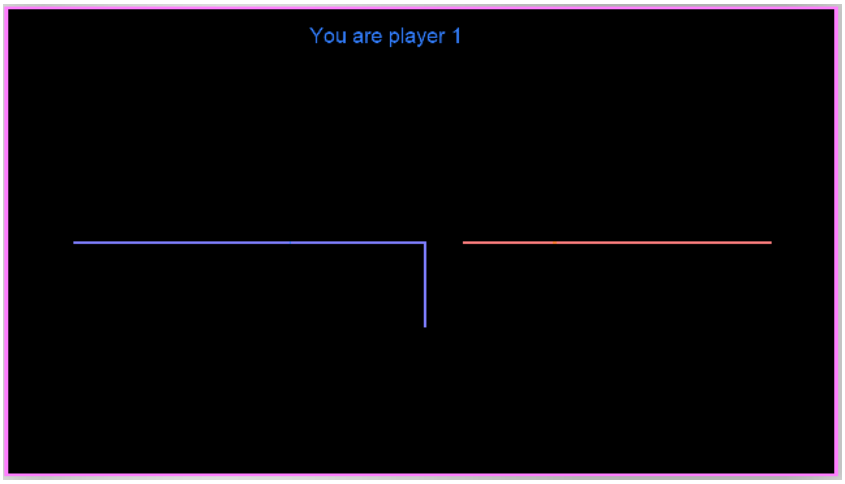

Figure 8.3: Game state for local-lag before rollback

In low latency situations, this solution adds unnecessary input delay. In cases with latencies higher than the local-lag amount, it will give more correct estimates for opponent positions than no delay. The downside is that it will rollback and fast-forward for every action received in order to keep the state correct, leading to more warps. Any jitter that still gets the messages there on time is invisible to the local player, which is a big strength of this solution.

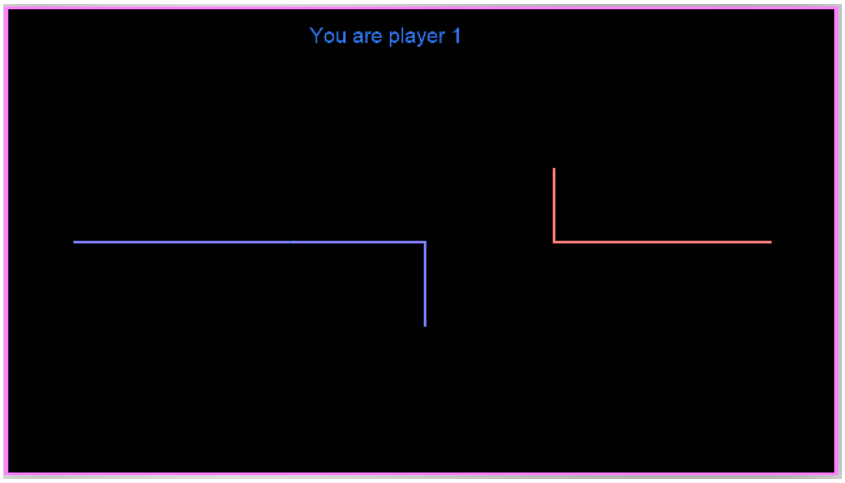

Figure 8.4: Game state for local-lag after a rollback

This implementation is a P2P solution since there is no absolute game state, and thus no need for a trusted server.

8.3 Implementation 3: Authoritative server

This implementation trades responsiveness for guaranteed correctness. The server holds the game state and performs all calculations. This means that the server reports if a player collides with a wall and finds out when and where a turn is applied. Players simply mirror the server state and send requests each time they want to perform an action. The server then applies the action when it receives the request, and broadcasts it including the time it was applied. This gives the client a input delay of 1/2 RTT, but 1 RTT before the client see the new game state.

Clocks are synchronized at the start of the game, and the server is ahead of the players by 1/2 RTT. If messages are received too early, they are queued and executed at the intended time. If they are received too late, the client rolls back, applies it and fast forwards to the current time.

Jitter for messages to the server for own actions will cause the input delay for the player to vary. This makes it hard for the player to predict when a turn will happen. Figure 8.5 and Figure 8.6 shows the turn happening at two different places for the same click, while Figure 8.7 shows a case where the message does not reach the server in time and the player collides with the wall and loses.

Jitter for messages traveling to the client causes warping as the client must roll back and then fast forward to the correct state.

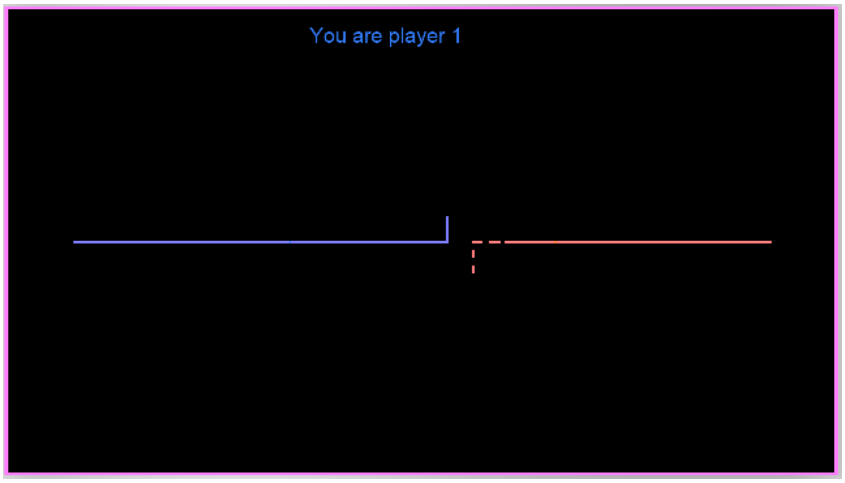

Figure 8.5: Game outcome for authoritative server with minor delay

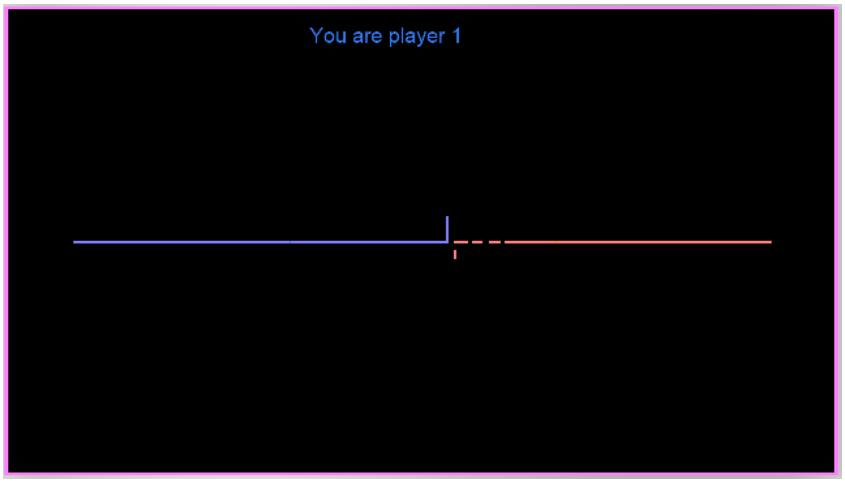

Figure 8.6: Game outcome for authoritative server with some delay

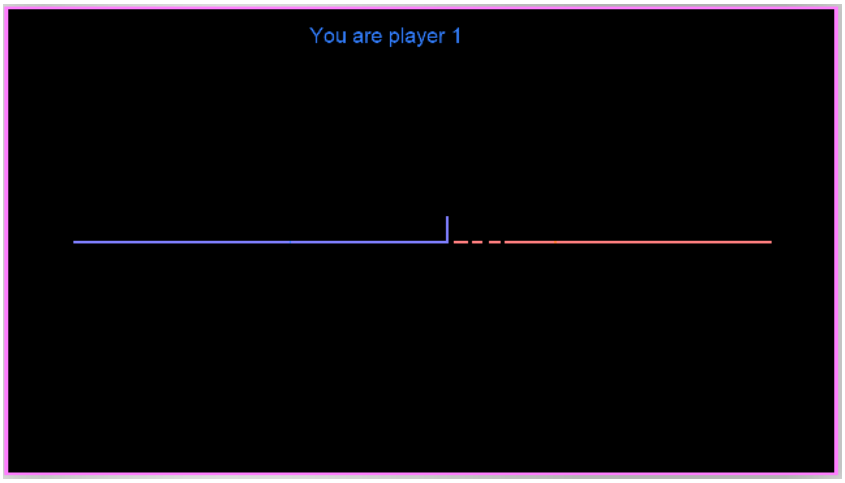

Figure 8.7: Game outcome for authoritative server with major delay

Chapter 9

User experiment

This chapter describes the experimental part of this project where the implementations suggested in Chapter 8 are tested and compared.

9.1 Objective

The goal of the experiment is not to compare the suggested implementations directly, but to compare them with a perfect implementation featuring no warps, no input delay and all users seeing the same game state. The suggested implementations are evaluated on how they compare to the perfect implementation where the goal is for them to be indistinguishable.

9.2 Testing method

Subjects test the perfect implementation and one of the three suggested implementations from Chapter 8. The perfect implementation is achieved through an offline mode where the user plays against an AI opponent. The user is not made aware that this is happening offline and the test setup suggests that it is versus a human opponent. This is done because if the subject was clearly playing a local game this could change the way he/she compares the two experiences.

9.2.1 Setup

The tests are performed in a closed room with two test operators. The first operator is in charge of explaining rules and observing the subject. The second operator serves as the opponent for the second part of the test, while pretending to be the opponent in the first part.

A test subject enters the room and the first operator explains the rules, using a script to ensure all subjects get the same instructions. The script used is available in Appendix A.

The subject is then given a device and asked to start the first version of the game. The subject plays this version for two minutes, before playing the second version for two minutes.

After playing both versions, the subject is asked to answer a set of question. The questionnaire contains both quantitative and qualitative parts. For both versions, subjects are asked about the user experience and then asked to compare the two. At the end, there are a number of optional questions where subjects can elaborate on topics of their choice. The questions asked is available in Appendix B.

9.2.2 Network conditions

To have full control over network conditions, all tests are run on WLAN using TCP and a dedicated server connected to the same network as the devices.

According to the tests in Chapter 5, WLAN adds an average of 15 ms per round-trip with spikes of up to 150 ms. To simulate proper network conditions, the dedicated server adds latency to all packets, allowing easy simulation of different network conditions. Since the tests showed spikes and jitter on WLAN connections, no extra jitter was added by the server.

Latency values The experiment uses network latency values that simulate real conditions. The tests in Chapter 5 show that EDGE networks are not suited for this type of game. Both HSPA+ and LTE networks feature RTTs of just under 100 ms, so the experiment uses 100 ms as RTT before accounting for distance.

As described in Section 3.1 it is fair to assume distance based latencies of close to 50 ms if sufficient servers are used or geographical matchmaking is strict enough. This test uses 75 ms to add some flexibility for the results. This allows matchmaking with most players on the same continent.

In summary, the experiment uses RTTs of 175 ms, based on 100 ms overhead from LTE/HSPA+ and 75 ms from distance. No delay (Section 8.1) and local-lag (Section 8.2) are simulated as peer to peer implementations. This means that the server adds 1/2 RTT for every packet. On the other hand, authoritative server (Section 8.3) simulates a dedicated server implementation. The server adds 1/2 RTT for inbound and outbound packets totaling 1 RTT. Based on the findings in Section 3.2, a 150 ms input delay should be noticeable in a negative way for players. We have chosen to use this value for this implementation, as it is a realistic scenario if server validation is needed. It is also interesting to see how the users respond to these high values.

Test subjects The test uses 18 test subjects, 6 for each of the implementations. The subjects for the test are friends, family, and coworkers of the authors. The gaming experience of the subjects varies greatly and age is spread from 25 to 54. There are 10 males and 8 females. The subjects are spread across the implementations as evenly as possible with regard to age, sex and gaming experience.

9.3 Results

This section contains the results from the user experiments. The results from the subjects testing the no delay implementation from Section 8.1 are gathered in Table 9.1. The results from the local-lag implementation (Section 8.2) are displayed in Table 9.2, while the authoritative server (Section 8.3) results are found in Table 9.3.

Each row in the table contains a question the subjects answered after completing the test. Subjects were asked to answer each statement on a scale from 1-5 with 1 being "I strongly disagree" and 5 being "I strongly agree". The exception is the "Total grade" question which ranges from "Very poor" to "Very good"

We have computed the average value and the standard deviation for each answer. In the rightmost column labeled A-B, we have subtracted the answer for version B from that of version A, giving an indication of how much of a difference users are able to notice.

9.3.1 Results no delay

This implementation performed well. As seen from Table 9.1, A-B is low for all questions which indicates that users experienced the two as similar. "The two versions felt the same" got an average score of 3.83, which is also good. This is as expected since the controls behave the exact same way for the local player. The difference is in how the opponent behaves, which is less noticeable for short play sessions.

Some test subjects preferred the no delay implementation to the offline, perfect implementation. It is not better in any technical way, but since the users tested this last they might have improved their mastery of the game which might lead to a better experience.

	A		B		A - B	
Question	Avg	σ	Avg	σ	Avg	σ
I receive immediate feedback on my actions	4.5	0.83	3.83	1.17	0.67	1.21
I feel the game treats me fairly	4.5	0.83	4.0	1.26	0.5	0.83
I feel my opponent has an advantage	2.5	1.76	2.67	1.63	-0.16	0.75
I find it easy to avoid the walls	3.16	1.17	2.83	0.75	0.33	0.81
Total grade	4.0	0.89	3.5	0.83	0.5	1.22

	A	B
My favorite version	66.6%	33.3%

	Avg	σ
The two versions felt the same	3.83	1.47

Table 9.1: Results for no delay

9.3.2 Results local-lag

The local-lag version scored very poorly in most questions. The total grade of 1.67 is by far the worst of the three tested implementations. It is also interesting to note that the total grade for version A, the offline implementation, is the lowest for the local-lag test at 3.5. The reason could be that a poor experience with version B pulls down the experience for the entire test.

The A-B column shows that subjects were clearly able to differentiate between the two versions, and this was clear from watching them during the experiment as well.

The score of 2.33 for immediate feedback on actions is surprising given our research on player tolerance in Section 3.2. An input delay of 100 ms should not be very noticeable, but when directly compared to 0 ms it is a lot more visible.

No player preferred local-lag to the offline version and the average score of 1.83 for "The two versions felt the same" shows that the two were significantly different.

	A		B		A - B	
Question	Avg	σ	Avg	σ	Avg	σ
I receive immediate feedback on my actions	4.67	0.51	2.33	1.36	2.33	1.36
I feel the game treats me fairly	4.16	1.17	2.5	1.64	1.67	1.50
I feel my opponent has an advantage	1.67	1.21	3.16	1.47	-1.5	1.04
I find it easy to avoid the walls	2.67	1.36	1.16	0.41	1.5	1.22
Total grade	3.5	1.05	1.67	0.51	1.83	1.16

	A	B
My favorite version	100%	0%

	Avg	σ
The two versions felt the same	1.83	1.60

Table 9.2: Results for local-lag

9.3.3 Results authoritative server

This test gave polarizing results. An example of this is shown in Figure 9.1. Some test subjects stated that it was unplayable and that the delay was way too high.

Two test subjects stated that they did not notice any difference between the two versions in this test, which is surprising. When observing them they compensated for the added delay quickly, but did not seem aware that they were doing it.

The average results does not represent the opinions of any of the test subjects, which makes them a poor basis for making claims.

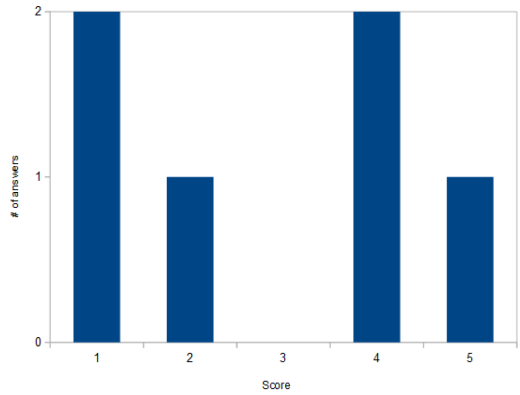

Figure 9.1: Authoritative server - I receive immediate feedback on my actions

	A		B		A - B	
Question	Avg	σ	Avg	σ	Avg	σ
I receive immediate feedback on my actions	4.83	0.41	2.83	1.72	2.0	1.89
I feel the game treats me fairly	5.0	0.0	3.16	1.47	1.83	1.47
I feel my opponent has an advantage	1.5	0.83	2.0	1.54	-0.5	0.83
I find it easy to avoid the walls	2.83	0.98	1.67	0.51	1.16	1.16
Total grade	4.33	0.81	2.83	0.98	1.5	1.37

	A	B
My favorite version	100%	0%

	Avg	σ
The two versions felt the same	2.33	1.75

Table 9.3: Results for authoritative server

9.4 User comments

Most subjects answered the optional questions at the end of the questionnaire. The most common comments were fast response for the offline version and slow response for the online version. Another common comment was that the control scheme was difficult to understand. The instructions did not specify that controls are relative to the current direction of the player.

Some players reported input delay for the no delay version even though both versions they had played had no input delay. This might indicate that subject are looking for a difference because they are faced with the direct question of what the difference is.

For local-lag and authoritative server, subjects commented that the game or the opponents were ”lagging”. This can mean a number of things, but is most likely referring to warping which did happen for some subjects.

Chapter 10

Evaluation

This chapter contains our evaluation of the latency test performed in Chapter 5, the implementations of Don't Touch The Walls described in Chapter 8, and the user experiment performed in Chapter 9. Our experiences from designing the game, implementing different solutions, observing the experiments and the user feedback from the experiments form the platform for evaluation.

10.1 Evaluating latency test

Performing the latency test was an interesting experience. The test compares itself to the 2009 test of Wang et al. [1] and shows dramatic improvements in round-trip times. Many things have changed since their test was performed and the devices used for testing are much more powerful. The mobile networks have changed. EDGE technology has been upgraded since 2009, and UTMS has been upgraded with HSPA and HSPA+, so improvements here can not be attributed solely to the devices.

Comparing the networks to one another is not that important. The results are interesting because the network latencies of today are lower than the network latencies of 2009. When mobile networks give average round-trip times of less than 100 ms, developers are able to create real-time multiplayer games with less synchronization issues. With the results from 2009, many latency handling mechanisms are unusable because of the high latency. One example being techniques where actions are validated by a server first.

TCP seems to be performing much closer to UDP than in the test from 2009. UDP still performs better and should be used whenever guaranteed delivery and order is not required. In the remaining cases, TCP looks like a promising alternative.

These tests were performed in a large city in Norway on high-end Android devices. It would be interesting to compare these to tests with low-end devices, other operating systems, more network types and networks in other countries. Even though our results suggest that players can get < 100 ms latencies, developers can

not assume that all players will be able to play under these conditions.

10.2 Evaluating implementations

This section evaluates the implementation of Don't Touch The Walls (DTTW). The main focus when choosing the game type was to find a simple yet challenging game where network issues were easy to spot. The game was constructed in order to highlight inconsistencies if they happen. It is worth noting that this is the exact opposite of what one would do when releasing a real game.

We chose a very simple graphic style, so that there were as few things as possible to distract the player. The tester always played as the same color and started at the same spot. There is only one opponent in a game where there could easily be more than one. This helps the player notice all the things that are happening since there are only two objects moving at a time.

We chose to keep the speed of the game high as this will make inconsistencies more visible. A rollback of 100 ms will move a significant piece of wall and the operation is quite visible.

By reducing the speed of the game, testers could have focused more on the game state as it would be simpler to avoid obstacles. The downside to reducing the speed is that inconsistencies would be less visible. If players have lower speed, they will be extrapolated shorter in the wrong direction before the state is corrected. In many cases, this might be enough for the player not to notice at all.

Two players experienced a bug with the offline version where the player is able to cross paths with the AI without anyone dying by moving into the same tile at the same time. This likely changed their opinion of the perfect implementation. Ideally, the game should have been tested more for bugs before the experiments and preferably on external testers.

Some players had issues with the control scheme because turning is relative to your current direction where games such as Snake used one key for each of the four possible directions. Doing some more iterations on the controls could have made the game itself easier to play, and allowed the testers to focus on the variations of the implementations they played.

10.3 Evaluating user experiments

This section evaluates the user experiments and looks at some of the results based on the choices made.

All test users played the perfect offline mode first. This meant that this was their baseline experience for input and correctness. The input delay on the local-lag and the authoritative server version were therefore highlighted as the game did not respond as the user had just gotten used to. If the players had tested this version first their experience might had been different. These users had to relearn the response time of the game and several subjects were vocal about their frustration.

Users that played the no delay version had the same control feel as the offline version. This made it easier for them to perform better in this version, and could be some of the reason that the no delay version was rated so much better than the other two.

The users were only given two minutes to play each game mode. If they had been given more time they might become more aware of the pros and cons of the different versions. In the first games the user often focused on understanding the controls and not the response and correctness of the game. The reason for choosing a short session length was partly that the game is so simple and fast paced that subjects feel "done" after a short time. Each game lasts from 5-30 seconds depending on how good the players are. By keeping the sessions short the subjects stay focused and do not have time to get bored.

The test was run with 18 test subjects spread across three different versions, leaving 6 persons per version. This means that each tester will significantly influence the result for that version. The testers had different amounts of gaming experience, and having a higher number of testers would make it possible to differentiate on age or gaming experience for each game implementation.

From observing the tests, we gathered that players are more affected by input delay than by incorrect game states. You die more often from unresponsive controls than from incorrect game states, so this is natural.

The wireless network used when conducting the tests misbehaved at times, and some test subjects had better experiences than others simply due to network conditions. This could be due to issues with the router or that there are many other networks in the area disturbing the signal. While this is not optimal for this test, it is a realistic scenario. When implementations rely on stable network conditions, it is interesting to run tests under unstable conditions. A larger sample size would even out the impact of such variations.

10.4 Evaluating results

The results from the user experiment show that the no delay version scored the best by a large margin. The two others are closer with authoritative server beating local-lag slightly. The key results from Section 9.3 are summarized in Table 10.1. The results from the initial questions show a larger difference than the direct question as summarized in Table 10.2. This could be due to the structure of the questionnaire. When faced with the direct question, users might feel inclined to say that they noticed a difference regardless of if they did. Some subjects who told us that they did not notice any difference have still answered this question with something other than the score of 5.

As discussed in Section 10.3 the overall results might have much to do with the way the user experiments were performed. Zero input delay makes the game feel more responsive and allows the user to respond to changes in the game quicker, reducing collisions with walls. Players seemed to enjoy the game more when they did not die all the time. This gives the no delay implementation an advantage.

In both the local-lag and authoritative server versions, the user must think

	No delay	Local-lag	Auth. server
Question	Avg	Avg	Avg
I receive immediate feedback on my actions	0.67	2.33	2.0
I feel the game treats me fairly	0.5	1.67	1.83
I feel my opponent has an advantage	-0.16	-1.5	-0.5
I find it easy to avoid the walls	0.33	1.5	1.16
Total grade	0.5	1.83	1.5

Table 10.1: Comparison of results (A-B average score)

	No delay	Local-lag	Auth. server
The two versions felt the same	2.33	1.75	2

Table 10.2: The two versions felt the same

further ahead when planning a action. The authoritative server has a variable delay which can make this very hard and we assumed that this would make it feel significantly worse. If the user simply turns at random times, and far from walls they might not notice this at all. Two of the users that played authoritative server felt that the version was practical identical which indicates that they did not pay attention to input delay.

We expected local-lag to perform better then authoritative server. Both version scores poorly on "I receive immediate feedback on my actions", but the difference between 100 ms and 150 ms does not seem that significant to the users. The important part is the change from 0 ms to > 0 ms latency. A lower delay is still better, but given the short play session the difference in input latency was not a huge factor.

The no delay version performs exactly as the offline version except in case of jitter or in cases where the players are close to each other. This is explained in Section 8.1. The authors played DTTW enough for these situations to become frequent. Experienced players tend to not die to random walls, but try to trick the other player into dying by cutting in front of them. This was not our favorite version, but we assumed that it would perform well in the test.

The authors preferred the local-lag version after having played DTTW a while. This version gave the best experience for competitive gameplay when a certain skill level was achieved. Since the input delay was acceptably small, this was the version the authors expected to perform the best in the tests.

The authoritative server issues with variable controls gets more visible the longer a user plays, and it looks like a 2 minute session was not long enough for this to happen. That this version scored the same as local-lag was a surprise. The input delay was also large enough that the authors disliked playing this version.

Part V

Summary

Chapter 11

Conclusion

Even though mobile networks continue to develop, the latency on the newest mobile networks are still much higher than what can be achieved on a LAN or WLAN. TCP and UDP have similar latencies on 3G and 4G networks, but TCP has more problems with jitter in congested network with package loss. On 2G networks, UDP performs much better than TCP on average.

Most of the user base worldwide is restricted to 2G networks. If a developer targets the 3G network, a latency of 100 ms should be expected. If the goal is to reach as many players as possible, a latency of 150-200 ms must be supported and UDP should be used. It is smart to keep data size down because users have limited bandwidth and many pay for the amounts transferred.

Our experiment found that a input delay of 100 ms compared to 0 ms was significant and noticeable by the users. Our no delay implementation of Don't Touch The Walls (DTTW) scored the best, but this could be because the response time was the same for both versions that subjects tested. The user did not play long enough to reach a high skill level where the incorrect game state made the game unfair. Most of the deaths in the game were collisions with outer walls and old player walls.

Since there are so many mobile games available through the app stores it is important to understand the importance of first impressions. It might be worth it to reduce input latency to improve the user experience for new players and let experienced players learn to work their way around any inconsistencies. Users could easily change to a different game which has no input latency if it feels better.

When designing network implementations it is important to focus on the parts that are visibly unfair for the users, such as a player starting before the others in a racing game. Every genre and every game has unique challenges and restrictions, and it is up to developers to identify the most important points and optimize these for their players. Some games require strictly correct game states, while others can easily cover up inconsistencies. Some are very sensitive to input delay, while others can tolerate larger amounts.

11.1 RQ1 Latency in modern networks with modern devices

The tests performed in Chapter 5 show that modern 3G and 4G networks can achieve round-trip times of less than 100 ms. TCP and UDP perform similarly with UDP having a slight edge. WLAN is still better than mobile networks by a large margin.

11.2 RQ2 Impact of latency

Latency impacts gameplay either by introducing inconsistencies or input delay. Latencies of less than 100 ms allows developers to use sophisticated mechanisms for synchronizing game state while keeping input delays at or below 100 ms. This is discussed in Section 3.3.

11.3 RQ3 Optimal implementation

The optimal implementation is dependent on the game itself, and there is no general answer for this question. The topic is discussed in Chapter 7, and each of the subquestions RQ3.1 to RQ3.4 are discussed in detail in their own sections in that chapter. One of the main takeaways is that optimizing for responsiveness and optimizing for correctness are conflicting goals, and developers need to carefully decide what is appropriate for the game in question.

11.4 RQ4 Best combination for user experience

As with RQ3, this question does not have one single answer and depends on the game in question. In general it is important to keep the game as responsive as possible while avoiding inconsistencies that users experience as unfair. Not all inconsistencies are noticeable, and not all of them have a large impact on user experience. One example is opponents moving in areas which the player does not see.

Responsiveness is most important for time-sensitive actions. An example from DTTW is turning, since slow responses can send the player head first into a wall. In implementations with input delay this can be masked by adding animations or unit responses that start immediately when the player performs an action, while the action is actually performed a few frames later.

Most real-time multiplayer games for mobiles can and should reduce latency by segmenting the user base on geographic location.

Chapter 12

Further Work

The experiment performed in Chapter 9 suffers from a small sample size. It would be interesting to perform a similar experiment with more test subjects to have more data to draw conclusions from. In particular, it would be valuable to separate subjects by gaming experience and see how this changes the results.

We would like to run the test with longer play sessions and see how this affects the result. It could also be interesting to have two user groups test the game with different session lengths and see if this changes things.

The way the experiment was structured likely favored the no delay version. It could instead be done by having each user play only one version and comparing the scores when the user has nothing to compare the experience to directly. Alternatively, the users could test all three online versions but not the offline version.

Finally, it would be valuable to test different levels of input delay directly. By having each user test the offline version with an added input delay and testing every 10 ms from 0 - 100 ms on a different set of users, one could see where the drops in user experience are the largest and what levels of input delay are optimal.

Appendices

Appendix A

Test instructions

The following instructions were given to test subjects. As the tests were performed in Norwegian, the script is also in Norwegian.

Hei, Du skal få teste to ulike varianter av vårt nyeste spill, Don't Touch The Walls. Spillet er en slags multiplayer versjon av snake. Du beveger deg fremover automatisk og kan svinge til høyre eller venstre. Klikk på venstre del av skjermen for å svinge til venstre, høyre del av skjermen for å svinge til høyre. Når du flytter deg legger du igjen en vegg bak deg. Man dør hvis man kræsjer i egne eller andres vegger, i tillegg til kantene på brettet. Sistemann i live vinner. Du kommer til å spille som blå farge og starte på venstre siden av brettet.

Du skal få spille begge variantene i 2 minutter og så skal du svare på noen spørsmål om opplevelsen.

Appendix B

Questionaire questions

The following questions were answered by test subjects. As the tests were performed in Norwegian, the questions are also in Norwegian.

Info
- Kjønn
- Alder
- Spillerfaring? (Casual, Gamer, Hardcore gamer)
- Hvilken variasjon B spilte du?

Variasjon A: (1-5, Svært uenig - Svært enig)
- Jeg får umiddelbar tilbakemelding på mine handlinger
- Jeg føler at spillet behandler meg rettferdig
- Jeg opplever at motstanderen min har en fordel
- Jeg synes det var lett å unngå veggene
- Totalkarakter for denne variasjonen (1-5, Svært dårlig - Svært bra)

Variasjon B: (1-5, Svært uenig - Svært enig)
- Jeg får umiddelbar tilbakemelding på mine handlinger
- Jeg føler at spillet behandler meg rettferdig
- Jeg opplever at motstanderen min har en fordel
- Jeg synes det var lett å unngå veggene
- Totalkarakter for denne variasjonen (1-5, Svært dårlig - Svært bra)

Sammenligning:
- Jeg likte best (A eller B)
- De to variasjonene føltes like (1-5, Svært uenig - Svært enig)

Tekstspørsmål:
- Hvis du merket forskjell mellom A og B, beskriv forskjellen?
- Hva var bra med variasjon A?
- Hva var bra med variasjon B?

Hva var dårlig med variasjon A?
Hva var dårlig med variasjon B?
Flere kommentarer?

Bibliography

[1] Alf Inge Wang, Martin Jarrett, and Eivind Sorteberg. Experiences from implementing a mobile multiplayer real-time game for wireless networks with high latency. *Int. J. Comput. Games Technol.*, 2009:6:1–6:14, January 2009.

[2] Dota 2 official website. `http://dota2.com`. Accessed 12.02.2015.

[3] Dota 2 server information and discussion. `http://dev.dota2.com/showthread.php?t=63789`. Accessed 07.02.2015.

[4] G. Armitage. An experimental estimation of latency sensitivity in multiplayer quake 3. In *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, pages 137–141, Sept 2003.

[5] Quake 3 on wikipedia. `http://en.wikipedia.org/wiki/Quake_III_Arena`. Accessed 12.02.2015.

[6] Startping.com - ping online from multiple different locations worldwide. `http://startping.com`. Accessed 07.02.2015.

[7] Justin Manweiler, Sharad Agarwal, Ming Zhang, Romit Roy Choudhury, and Paramvir Bahl. Switchboard: A matchmaking system for multiplayer mobile games. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 71–84, New York, NY, USA, 2011. ACM.

[8] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '02, pages 23–29, New York, NY, USA, 2002. ACM.

[9] Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Commun. ACM*, 49(11):40–45, November 2006.

[10] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The effect of latency on user performance in warcraft iii. In *Proceedings of the 2Nd Workshop on Network and System Support for Games*, NetGames '03, pages 3–14, New York, NY, USA, 2003. ACM.

[11] Blizzard entertainment: Warcraft 3. http://us.blizzard.com/en-us/games/war3/. Accessed 12.02.2015.

[12] Laurent Gautier, Christophe Diot, and Jim Kurose. End-to-end transmission control mechanisms for multiparty interactive applications on the internet. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1470–1479. IEEE, 1999.

[13] Lothar Pantel and Lars C. Wolf. On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st Workshop on Network and System Support for Games*, NetGames '02, pages 79–84, New York, NY, USA, 2002. ACM.

[14] Yahn W Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference*, volume 98033, 2001.

[15] Ouri Wolfson. The overhead of locking (and commit) protocols in distributed databases. *ACM Trans. Database Syst.*, 12(3):453–471, September 1987.

[16] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg. Local-lag and timewarp: providing consistency for replicated continuous applications. *Multimedia, IEEE Transactions on*, 6(1):47–57, Feb 2004.

[17] Abdul Malik Khan, Sophie Chabridon, and Antoine Beugnard. A dynamic approach to consistency management for mobile multiplayer games. In *Proceedings of the 8th International Conference on New Technologies in Distributed Systems*, NOTERE '08, pages 42:1–42:6, New York, NY, USA, 2008. ACM.

[18] Vijay Garg. *Wireless Communications & Networking*. Morgan Kaufmann, 2010.

[19] All about the technology. http://www.itu.int/osg/spu/ni/3G/technology/. Accessed 14.02.2015.

[20] About mobile technology and imt-2000. http://www.itu.int/osg/spu/imt-2000/technology.html. Accessed 14.02.2015.

[21] Itu global standard for international mobile telecommunications imt-advanced. http://www.itu.int/ITU-R/index.asp?category=information&rlink=imt-advanced&lang=en. Accessed 14.02.2015.

[22] Itu facts and figures 2014. http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2014-e.pdf. Accessed 07.02.2015.

[23] Ericsson network traffic. http://www.ericsson.com/TET/trafficView/loadBasicEditor.ericsson. Accessed 07.02.2015.

[24] Apple unveils iphone. http://www.macworld.com/article/1054769/iphone.html. Accessed 14.02.2015.

[25] Armagetron advanced on wikipedia. `http://en.wikipedia.org/wiki/Armagetron_Advanced`. Accessed 12.02.2015.

[26] Tron on wikipedia. `http://en.wikipedia.org/wiki/Tron`. Accessed 12.02.2015.

[27] Corona game engine. `http://coronalabs.com/`. Accessed 08.02.2015.

[28] De-Yu Chen, Po-Ching Lin, and Kuan-Ta Chen. Does online mobile gaming overcharge you for the fun? In *Network and Systems Support for Games (NetGames), 2013 12th Annual Workshop on*, pages 1–2, Dec 2013.