Minimising Lag in Game Networking Models Without a Central Server

Implementation and Analysis of Client-Hosted and Peer-to-Peer Networking Models in the Context of Games.

Szymon Jackiewicz
Student Number: 150249751
Supervisor: Dr Graham Morgan

Word Count: ???

MComp Computer Science w Games Engineering
Stage 3

Department of Computer Science
Newcastle University
5 March 2019

Declaration

"I declare that this dissertation represents my own work except, where otherwise stated."

Acknowledgements

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Minimising Lag in Game Networking Models Without a Central Server

Implementation and Analysis of Client-Hosted and Peer-to-Peer Networking Models in the Context of Games

Szymon Jackiewicz

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Contents

Chapter 1: Introduction

Throughout this project, I will be exploring the different methods of providing a synchronised, multiplayer gaming experiance on two or more computers on a network. I will be covering the networking basics of how two game clients running on seperate machines can communicate with each other as well as exploring the relevant details of how and why the networking systems in games are designed the way they are.

I will explore the existing methods of writing an online multiplayer game from scratch and aim to provide an implementation of a networking library for online games to communicate without a central server. I also aim to provide analysis of efficieny of different methods of achieving this goal.

1.1 Basic concepts and terminolory

First of all, I will define some basic concepts and terminolory that will be used throughout this document.

Client: Within the context of this document, a client can be defined as a piece of software responsible to running the game (i.e. game client). A client can also be defined as a computer interacting with a server, however it is possible to run two different instances of a game client on a single machine.

Online Multiplayer Game: A video game can be defined as a simulation of a certain scenario that can be manipulated by the player of the game. When talking about online multiplayer games, it can be thought of as a simulation that runs on several clients connected by a network (e.g. LAN or the Internet) that is to be synchronised. When one player performs an action that effects the state of the simulation, this action should also be seen by all participants of this perticular simulation instance and therefore the simulation should remain in the same state across all participating clients.

Ping: In network connections, the ping between different clients refers to the shortest amount of time that is needed for one client to send information to another and receive a response from this client. One client sends a "ICMP echo request" to another networked client (e.g. a game server). The receiving client, then responds with an "ICMP echo reply" back to the original device. The time between sending the request and receiving the reply, is the ping between the two clients.

Lag: The grater the ping between two connected clients, the bigger the difference in the state of each clients' simulation once an action to be synchronised is performed. When a change is made by one client, this change should be seen by other clients participating in the same simulation and lag occurs when this change does not appear instentaneous to the user.

Jitter: TODO

1.2 Networking principles in games

TODO: talk a bit about UDP, TCP/IP and routing...

1.2.1 The Centralised Server Model

Most online multiplayer games that are played today, make use of the "centralised server model" for sinchronising the simulation state between several clients participating in the same simulation. This means that in an example of a First Person Shooter (FPS), if one player presses the "jump" key, their character will jump and this information is would also be sent over to the game server. The server would then send the information that this player has jumped, to all other clients. There is a potential problem here however. Given that the ping between the server and client A is α and between the server and client B is β, the time between client A pressing an input and client B being notified of this input can not be less than $\alpha + \beta$ and due to the limitations of physics $\alpha > 0$ and $\beta > 0$. This means that at any given time the lag experianced between clients A and B will be more than $\alpha + \beta$ when processing times are factored in too. This leads to a problem of a poor experience for a client with a high ping to the server, as they will receive the updates from the server later than every other player and therefore be at a disadvantage if the game requires real time reactions. Unfortunately under some implementations, this also results in a poor experiance for every other player, who despite having resonable ping to the server, can be shot from behind cover by a laggy player who fired a shot before the cover was reached.

Possible solutions to the variable ping problem

TODO: talk about region locks, and disconnecting players with high ping but ping can be high even in normal conditions

Chapter 2: Research

TODO:

2.1 Existing networking implementations

TODO:

2.1.1 Variable ping system in Battelfield 1

An interesting approach to the issue of variable ping in a game has been implemented by DICE in the game Battelfield 1. Given 2 players; player A with a low ping to the server and player B with a ping of <150ms to the server.

When player B fires at a moving player A, player B's client will perform the check concluding that player A has been hit and this information is sent to the server. The server will then perform it's own checks and if the server agrees that this hit is possible, then it sends the hit confirmation to player B and damage information to player A. This approach is called Clientside-Server Authoritative as while the hit registration is calculated on clientside, the server must still confirm that this is valid.

Concidering another scenario, suppose that player A still has a low ping to the server but player B, now has the ping of >150ms. An icon will appear on player B's UI showing an "aim-lead" indicator. Now when the shot is fired in the same scenario, the hit will not register anymore as the hit registration has switched from Clientside-Server Authoritative to Fully-Server Authoritative, meaning that the check is performed only once the shot information is received by the server.

Whilest this implementation makes the game feel less responsive for players with high ping, it provides a lot more fairness for everyone else and allow for players with different pings play in a more fair way.

2.1.2 Networking in Apex Legends

Apex Legends is a "Battle Royale" game by Respawn Entertainment. Due to the genre of this game, the implementation of networking has been implemented in an interesting way. The premise of the game consists of a starting amount of 60 players, all competing to be the last one alive at the end. An interesting aspect of this, is that while at the start of a match, the server might have to work quite hard to effectively synchronise the simulation for 60 different clients, as more and more players die off and less are left, less information needs to be sent to each clients, freeing up some server performance.

Since transferring all the information about 60 different players would most likely exceed the average MTU of a UDP packet, with each update, the server sends 1 smaller packet per each player to each player. This means that when 60 players are in the game, 60 packets are sent to 60 different clients from the same server. This is shown in the Netcode analysis in *Apex Legends Netcode Needs A Lot Of Work*(Battle(non)sense).

An interesting outcome of this, is quite noticible network lag that slowly goes away, as less and less players remain in the game. It is also likely due to this, that Respawn felt the need to implement a "Client Authoritative" aiming model. Meaning that the server will favor what the client sees over it's own view of the simulation (i.e. If the client

claims that a shot has hit an enemy, the server is likely to agree). The choise for this implementation could have been made to fix two potential issues; reducing the load on the server by reducing the need to perform extencive checks for each shot (this could be significant if a lot of players are playing), making the game feel more responsive on slower client hardware that may need more time to process up to 60 packets that arrive from the server at each update tick.

2.2 Security

TODO:

Chapter 3: Design

3.1 Code Architecture

Test for inline: `test Hello;`

3.2 Protocols

During any connection that can be deployed between several processes on a machine or even different physical hardware in seperate geographical locaitons, many different protocols come into play.

3.2.1 Protocols in Network Communications

Firstly, in the network layer, most commonly the IP[1] is employed however other options such as X.25 are also available but have more niche uses. These protocols are responsible for "packaging" the data to be sent between two different computers identified by their IP address. The packet from the sending machine, will travel through a network of routers that will eventually lead it to the machine with the IP address of the recieving machine.

Next comes the transport layer where either UDP[2] or TCP[3] can be chosen, both have different properties, advantages, disadvantages, useses and both are used in game networking. The UPD protocol is a simple, connectionless protocol which will simply send a packet from one IP address to another. Since each packet sent with UDP, can take a different route through the router network, there is no guarantee that the packets will be received in the same order as they were sent in. Due to many different reasons, packet loss can occur, meaning that it also cannot be guaranteed that every packet sent with UDP will arrive at the destination at all. Despite these dissadvantages and due to the simplicity of how this protocol was designed with it's connectionless nature, it inately has a major advantage in the speed that the packets can just be sent out and forgotten about. The TCP protocol, is built upon UDP to add some important features for reliable data sharing at a cost of speen and use in real-time applications. The most important property of TCP includes the assurance that if a packet is not received by a recipient, it is requested to be resent to guarantee that every pacet that is sent, is also recieved. This also means that the packets are aranged in the same order that they were sent meaning that we can be sure that not only data will arrive at the destination, but it will arrive just as we sent it. This implementation has many obvious benefits and in most scenarios, the delay of possibly re-sending a packet if it was not received is neglegible. In real-time applications however, this is likely to be an unnecessary waste of time and resources as even id a packet is dropped and and resent, by that time, new updated information is available so resending the dropped packet is useless when a packet with new information could be sent at that time instead. Simply, old information is quickly outdated and it's more important to send new information then old, non-useful information.

[1] IP: Internet Protocol
[2] UDP: User Datagram Protocol
[3] TCP: Transfer Control Protocol

The next layer of protocols is the Operating System Iterface or library, that is called by applications needing to share data using the above protocols. With Windows, a library called "WinSock" is often used, however other options are also available such as enet, asio, RakNet... This is where a programmer would be able to configure which protocols to use (Like TCP or UDP, IP or X.25 amongst many other configurable options).

3.2.2 Message Structure Startegies

Data Representation

There are many different ways that the same data can be represented and each one is carefully designed to be the most appropriate for it's use. Consider the two figures below of common ways of goruping complex data in an "easily readable" format; XML in Figure 3.1 and JSON in Figure 3.2. These example have been adapted from the article *Reading and Writing Packets*(Fiedler).

```
<world_update world_time="0.0">
  <object id="1" class="player">
    <property name="position" value="(0,0,0)"></property>
    <property name="orientation" value="(1,0,0,0)"></property>
    <property name="velocity" value="(10,0,0)"></property>
    <property name="health" value="100"></property>
    <property name="weapon" value="110"></property>
    ... 100s more properties per-object ...
 </object>
 <object id="110" class="weapon">
   <property type="semi-automatic"></property>
   <property ammo_in_clip="8"></property>
   <property round_in_chamber="true"></property>
 </object>
 ... 1000s more objects ...
</world_update>
```

Figure 3.1: An example of a representation of world data in the XML format

```
{
  "world_time": 0.0,
  "objects": {
    1: {
      "class": "player",
      "position": "(0,0,0)",
      "orientation": "(1,0,0,0)",
      "velocity": "(10,0,0)",
      "health": 100,
      "weapon": 110
    }
    110: {
      "class": "weapon",
      "type": "semi-automatic"
      "ammo_in_clip": 8,
      "round_in_chamber": 1
    }
    // etc...
  }
}
```

Figure 3.2: An example of a representation of world data in the JSON format

Checking for Packet Loss in connection

The packet information could contain a certain amount of bits that would be incremented with each simulation step[4]. This counter value could loop round when a maximum value is reached as long as several simulation steps in a row have unique values. The receiver could evaluate this value when received, checking if a packet has been dropped since the last received update. Knowledge about the quality of the connection could be important information when determining how much of the simulation has to be estimated between the received updates and could also be vital information to the player when in a game demanding split-second reaction time allowing them to change their strategy with the knowledge that they may be at a dissadvantage against other players.

[4]A simulation step refers to a state of values that represent the current state of a simulation. Each time the values are updated, is a new simulation step. This is often done and broadcasted several times a second in central server models.

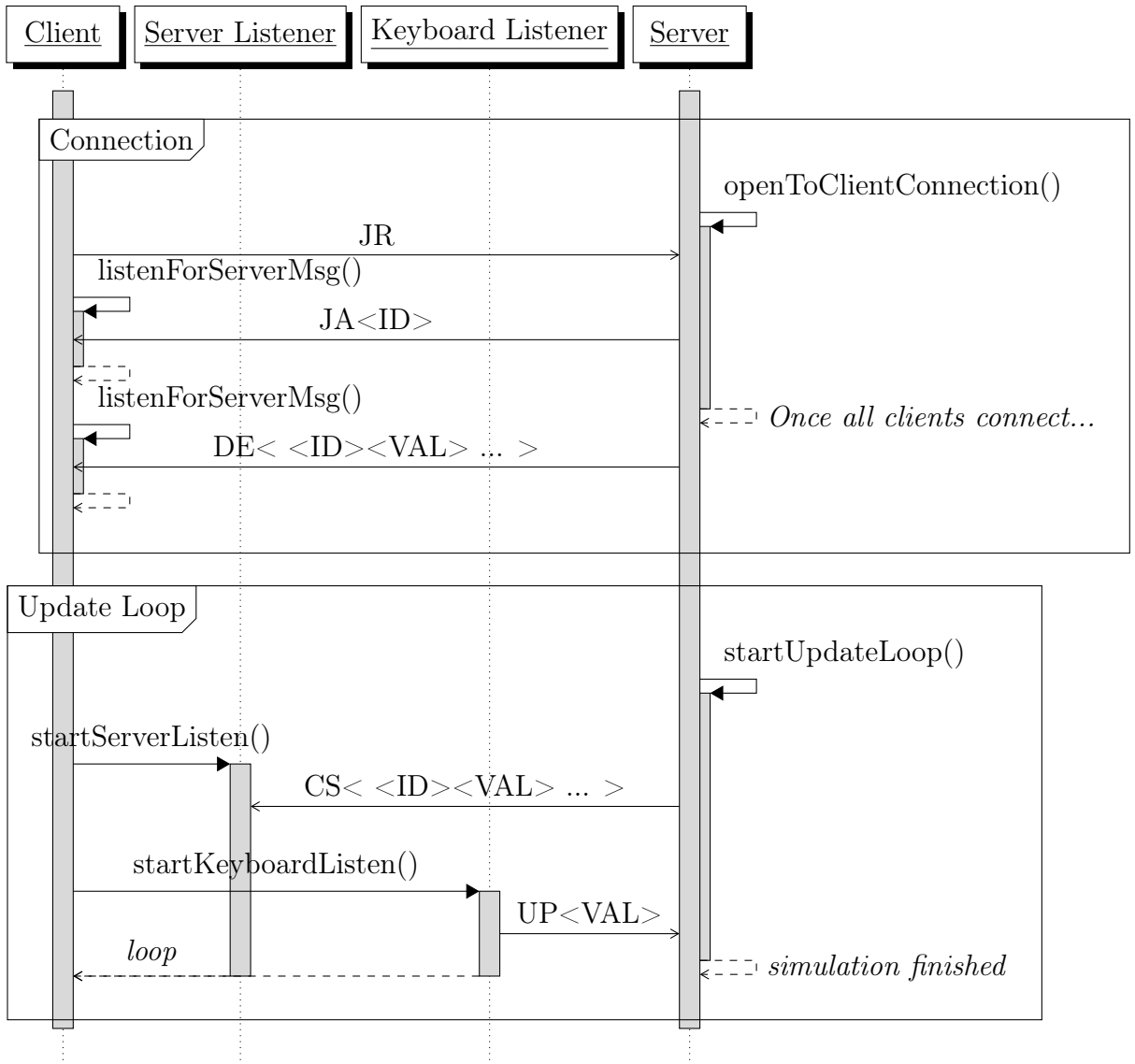

Figure 3.3: Graph showing the protocol of joining a session hosted on the server as well as sending and receiving updates.

3.2.3 Potential issues with the Client Hosted protocol

The protocol for establishing a connection and transfering of data can be found in Figure 3.3. There are many potential flaws with this approach.

Packet Loss

Time outs in place in case response lost... resend.... Resend request if no response....

Security

anyone could send update by spoofing ip

3.3 Message Codes

Message Type	Message Code	Description	Example Payload
Join Request	`JR`	Allows a client to send a join request to the server.	`JR`
Join Acknowledgement	`JA`	Allows the server to confirm that the client's information has been saved. Is followed by 1 byte indicating the client's ID	`JA1`
Ping Request	`PQ`	Message instructing the recipiant to reply with `PS`. Can be used to time the delay in this connection.	`PQ`
Ping Response	`RS`	This should be sent whenever a `PQ` message is received.	`RS`
Update	`UP`	Used by a client to update it's value on the server. Is followed by 1 byte representing the new value.	`UP9`
Define	`DF`	Used by the server to define the initial values for each of the clients connected to this instance. It is followed by a non-zero, even amount of bytes representing the client ID and it's value pair.	`DF1020304050`
Current State	`CS`	Used by the server to broadcast it's real state to all clients. When this is received, clients are expected to update their local state to this. It is followed by a non-zero, even amount of bytes representing the client ID and it's value pair.	`CS1927344157`

Table 3.1: Table showing the message codes for distinguishing messages from each other and how each one is to be used

Appendix A: Appendix Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

References

Battle(non)sense (2019). *Apex Legends Netcode Needs A Lot Of Work*. URL: https://www.youtube.com/watch?v=9PfFPW9a90w&t=2s.

Fiedler, Glenn (2016). *Reading and Writing Packets*. URL: https://gafferongames.com/post/reading_and_writing_packets/.