Bachelor's thesis

Information and Communications Technology

2022

Pasi Aaltonen

NETWORKING TOOLS PERFORMANCE EVALUATION IN A VR APPLICATION

– Mirror vs. Photon PUN2

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2022 | 62 pages

Pasi Aaltonen

NETWORKING TOOLS PERFORMANCE EVALUATION IN A VR APPLICATION

- Mirror vs. Photon PUN2

The concept of multiplayer was already formed in the 1970s. Throughout the years, the multiplayer concept has grown from locally played games on a single computer to games played through a local network to a globally reachable concept.

The purpose of this thesis was to study different tools for multiplayer games and evaluate possible differences in performance between them. This thesis project was commissioned by Ade Ltd. The first phase of the thesis focuses on multiplayer architecture and available multiplayer frameworks from which two were chosen for comparison, an open-source solution Mirror and software as a service solution Photon PUN2.

The evaluation of multiplayer solutions’ performance was carried out using the Unity game engine. The outcome of this thesis project was not a full VR training application, but the project was executed as a technology demo. The aim of this thesis was to research how user count increase affects the performance of an application and network traffic. The requirements for the multiplayer solution and metrics for performance and network traffic data collection were determined together with the commissioner.

The performance evaluation was executed in the FIT Turku Center's premises local network by collecting data from FPS stability, CPU and system memory usage, amount and size of sent packets through the network, and used bandwidth. The data analysis indicated Mirror to be more suitable for this type of VR application as with PUN2 there were noticeable delays in movement synchronization on lower user count than with Mirror.

Keywords:

Virtual reality, multiplayer, Unity, framework, Mirror, Photon

Opinnäytetyö AMK | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2022 | 62 sivua

Pasi Aaltonen

Moninpelityökalujen performanssiarvio VR -sovelluksessa

- Mirror vs. Photon PUN2

Moninpelikonsepti juontaa juurensa jo 1970-luvulta. Vuosien saatossa moninpeli on kasvanut samalla koneella pelattavasta moninpelista lokaaliverkon välityksellä pelattavaksi ja siitä aina internetin yli koko maailman kattavaksi konseptiksi.

Tämän opinnäytetyön tarkoituksena oli tutkia eri työkaluja moninpelin luomiseksi ja selvittää näiden mahdollisia eroavaisuuksia performanssin osalta. Työ toteutettiin yhteistyössä Ade Oy:n kanssa. Työn ensimmäisessä vaiheessa keskityttiin moninpeliarkkitehtuuriin sekä olemassa oleviin moninpelityökaluihin, joista lopulta vertailuun valittiin avoimenlähdekoodin ratkaisu Mirror sekä palveluratkaisuna tarjottava Photon PUN2.

Moninpeli ratkaisujen performanssiarvio toteutettiin käyttäen Unity -pelimoottoria. Työn lopputuloksena ei ollut kokonainen monin pelattava VR -sovellus vaan työ toteutettiin tekniikkademona. Tutkimuksessa kerättiin dataa eri käyttäjämäärien vaikutuksesta sovelluksen performanssiin sekä verkon kuormitukseen. Vaatimukset moninpelratkaisulle sekä kerättävälle datalle päätettiin yhdessä toimeksiantajan kanssa.

Performanssitutkimuksen datan keräys toteutettiin FIT Turku -osaamiskeskuksen tiloissa lähiverkon välityksellä, jolloin dataa kerättiin sovelluksen FPS:n tasaisuudesta, CPU:n sekä järjestelmämuistin käytöstä, siirrettävien pakettien määrästä ja koosta sekä kaistanleveydestä. Data-analyysin perusteella tämän tyyppiselle VR -sovellukselle Mirror näytti lopulta olevan parempi vaihtoehto sillä PUN2:lla esiintyy nähtävää viivettä liikkeiden synkronoinnin osalta alemmilla käyttäjämäärillä kuin Mirrorilla.

Asiasanat:

Virtuaalitodellisuus, moninpeli, Unity, viitekehys, Mirror, Photon

Contents

Figures

Pictures

Tables

List of abbreviations

Avatar	Digital model of a player-controlled character.
API	Application Programming Interface. A connection between computer programs.
CPU	Central Processing Unit. Computers brain that retrieves and executes instructions.
FPS	Frames Per Second. The number of images that are displayed on screen in one-second duration.
GB	Gigabyte. A memory unit.
HMD	Head-mounted display. A virtual reality headset device.
HUD	Head-Up Display. A display is used in games to show data to the player while playing the game.
KB	Kilobyte. A memory unit.
LAN	Local area network. LAN is a term used to describe computers that are connected within a small area.
LTS	Long-Term Support. A stable program release is maintained for a longer period.
P2P	Peer-to-Peer. A networking architecture model where multiple devices are connected to each other.
RPC	Remote Procedure Call. A method to send and receive data from other players or game servers in a multiplayer game.
SDK	Software Development Kit. A set of software tools provided by software vendors for developers to use in building software.

TB	Terabyte. A memory unit.
TCP	Transmission Control Protocol. A protocol to send data packets over the network.
UDP	User Datagram Protocol. A protocol to send data packets over the network.
Unity	A game engine by Unity Technologies.
Unreal	A game engine by Epic Games.
VoIP	Voice over Internet Protocol. Protocol to transfer audio such as speech over the network.
VR	Virtual Reality. A simulated environment to present life-like interactions with objects.

1 Introduction

The concept of a multiplayer game was formed already in the 1970s when a game called Pong was released to the consumer market. From there, the multiplayer concept has grown throughout the years from local multiplayer games played on the same machine to games played through a local network and from there to a global concept bringing players together from all over the world. Besides entertainment games, multiplayer technology can be utilized in learning applications and social platforms. Virtual reality can bring users closer together in a more immersive way.

This thesis focuses on the multiplayer aspect of virtual reality learning games by researching and comparing available multiplayer networking tools for the Unity game engine. The main objective of the thesis is to compare and evaluate the performance between open source and multiplayer as a service solution to evaluate performance in a VR game technology demo.

Chapters 2 and 3 present and discuss the available multiplayer solutions and the theory of networking architecture for multiplayer game development. These chapters cover different multiplayer topologies, server solutions, and transfer protocols.

Chapters 4 and 5 analyze and present the requirements for the VR training applications multiplayer solution and the core concepts of multiplayer solutions implementation to the Unity project using LTS Unity version 2020.3. Chapter 5 explains how the multiplayer solution was implemented.

Chapter 6 presents and discusses the performance evaluation of the two multiplayer implementations. This chapter explains research methods, metrics, and tools used in application performance, network traffic, and user experience evaluation as well as results in analysis outcomes and suggestions for further development.

2 Existing networking tools

A multiplayer framework is a set of tools provided to help developers to create multiplayer games without having to build a multiplayer solution from the start. (Neelakantam, 2021.) Currently, there are multiple different multiplayer frameworks available for developers to try and determine which would suit best for their game. Available options vary from free and paid open-source solutions which offer only the framework for multiplayer game development to multiplayer engines offering the whole multiplayer package as a complete service.

Open-source options give the freedom for the developers to choose the backend solution by themselves from available server service providers or to create a completely own networking solution. With multiplayer engine package services, developers are more locked to what service providers have to offer.

There is not a one size fits all type of solution when it comes to multiplayer, instead, developers have to think about what different multiplayer frameworks have to offer that would be the most suitable selection for their game. To help the decision-making process Unity has provided a review (Picture 1) to act as a guide by comparing some of the different multiplayer options by evaluating them in multiple different areas. Unity's survey study included 200 Unity user interviews and 20 in-depth interviews from Unity developers. Each multiplayer solution has its advantages and disadvantages making them suitable for different types of games. (Unity, 2021.)

	Stability/ support	Ease-of-use	Perfor-mance	Scalability	Feature breadth	Cost*	Customers recommend for
MLAPI	★★★★★	★★★★☆	★★★★★	★★★★☆	★★★★★	Free	Most client-server games for up to ~64 players that want a stable breadth of mid-level features
DarkRift 2	★★★★★	★★★☆☆	★★★★★	★★★★★	★★★☆☆	$100 for source	Games with high perf/ scale requirements that want to build on a stable LL layer
Photon PUN	★★★★☆	★★★★★	★★★☆☆	★☆☆☆☆	★★★★☆	$0.30/PCU	Simple and small (2–8 players) mesh-topology games
Photon Quantum 2.0	★★★★☆	★★★★☆	★★★★★	★★☆☆☆	★★★★☆	$1000/mo + $0.50/PCU	Games desiring deterministic roll-back, like MOBA games, for up to 32 players
Mirror	★★★★★	★★★★★	★★★☆☆	★★★★☆	★★★★☆	Free	Stable and proven client-server solution, loved best for its community and ease-of-use

Picture 1. Unity multiplayer review. (Unity, 2021)

This thesis focuses solely on multiplayer options for the Unity game engine. But there are also other game engines available for multiplayer such as Unreal Engine (UE) by Epic Games which provides its own multiplayer solution. There is also available a UE plugin called VR Expansion Plugin compatible with both UE4 and UE5 which provides tools for VR interactions and also for multiplayer and networking making it easier to start multiplayer game development with UE (Statzer, 2022).

2.1 Mirror

Mirror is one of the most used open-source multiplayer frameworks and many commercial games have been built using Mirror including titles like Population one, Zoomba, and The wall. It was created in 2018 for Unity to substitute the deprecated UNet multiplayer framework. For this reason, it has many similarities to UNet which makes the transition to Mirror much easier.

Mirror is a high-level networking library that is built on top of lower-level transport real-time communication layer with a server authoritative system that

also supports any kind of networking topology. This means that no dedicated server process is required and players can as clients and the server at the same time. (Mirror, 2022.)

Mirror offers support for Unity's LTS version, but also any version beyond should also work. It is highly recommended to use the LTS version for stability as it is not guaranteed that all features will work in non-LTS versions. (Mirror, 2022.)

2.2 Fish-Net

Fish-Net multiplayer framework is a rather new solution brought available to Unity networking. Its developer First Gear Games has stated that Fish-Net was created to be a competitor for Mirror.

Fish-Net is a free networking solution made for Unity. Same as Mirror, Fish-Net is designed to be server authoritative which allows users to act as a server and a client to ease the development. Fish-Net also supports any kind of network topology through its transport system which means that to allow communication between clients and servers transports can use a variety of technologies. (Fish-Net, 2022.)

On top of LTS versions of Unity 2019 and 2020, Fish-Net is currently compatible with Unity 2021. With non-LTS versions of Unity 2019 or 2020, all features should work but, are not officially supported. (Fish-Net, 2022.)

According to documentation, Fish-Net seems to be a solid competitor in the market, but time will tell how firmly this new framework will be supported and embraced by developers. At the time of writing the future for this solution seems promising.

2.3 Photon Engine

Photon offers different types of multiplayer game service packet solutions to choose from. These are Fusion, Quantum, PUN, and Bolt. Fusion is Photon's newest high-end netcode SDK for Unity which is made to merge the best concepts of Photon PUN and Bolt. It is recommended by Photon that Fusion is used for new projects instead of PUN or Bolt as these services are currently legacy but still supported. (Photon Engine, 2022.)

Quantum is a high-performance deterministic ECS (Entity Component System) framework for Unity online multiplayer games. Quantum is based on the predict/rollback approach which is most suitable for latency-sensitive online games like action RPGs, sports games, fighting games, and FPS games. (Photon Engine, 2022.)

PUN (Photon Unity Networking) is a Unity package to build multiplayer games. It is available for download in the Asset store with free and paid PUN Plus versions. PUN offers flexible matchmaking which gets players into rooms where objects are synced over the internet using dedicated Photon servers. PUN is designed to give an easy entry to multiplayer game development. The downside is that Photon PUN does not have the option for host-client multiplayer. (Photon Engine, 2022.) Photon PUN uses mesh topology, a direct peer-to-peer, where all players are connected to each other in a game room. This means that all players handle the data routed via the server by themselves. (Unity, 2021.)

Bolt offers client-hosted networking for Unity with the possibility to build multiplayer games with dedicated server architecture. The option to use self-hosted dedicated servers is unfortunately limited to paid subscriptions only. Bolt offers support to listen server or client-hosted where one client hosts the game, two-player peer-2-peer direct connection, and local LAN / WiFi with automatic host detection. (Photon Engine, 2022.)

Photons services vary between different packet solutions to give scalability based on game userbase growth. These packages start from a free trial service

using Photon cloud servers where the maximum player count is limited to 20 players per room and 3 GB traffic per player in a game session and with paid service goes up to 2000 connected players at the same time. Many games are using Photon multiplayer solutions like Outward, Runbow, and Drunkn Bar Fight (Photon Engine, 2022.)

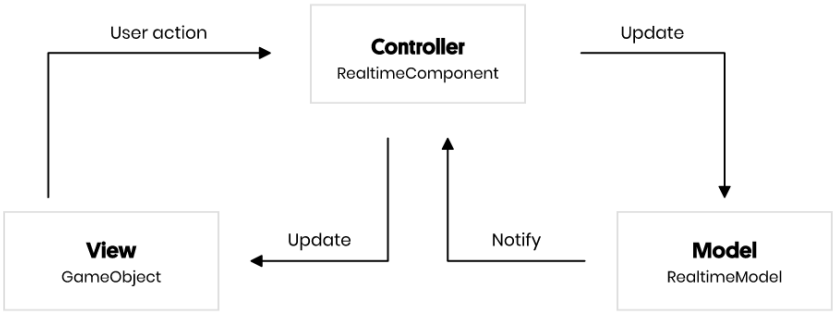
Although Photon is free to start using services it is not recommended to launch games for production using a free subscription plan. Free plans are feasible for fast concept prototyping and trying out different services.

2.4 Normcore

Normcore is a complete networking and hosting solution for Unity created by Normal. Normcore is a scalable hosted service that is built on a client-server model where the state of all clients in a room is kept synchronized. According to Normcore documentation, it is well suited for everything from mobile games to MMORPGs and from VR applications to productivity tools. (Normcore, 2022.)

Besides paid subscriptions, Normcore offers a free tier to quickly test their services. The free tier includes 30 concurrent users, 10 rooms, 50 room hours, and 120GB bandwidth. Paid tiers vary from Pro and Unlimited where the Pro tier offers unlimited concurrent user and rooms, 1000 room hours, and 3TB bandwidth. Unlimited tier offers all including to Pro tier plus also additional usage for an extra cost. (Normcore, 2022.)

According to documentation, Normcore uses Model, View, Controller (Picture 2) as their based architecture. This architecture helps establish a clear separation of concerns for what handles networking code. (Normcore, 2022.)



Picture 2. Normcore Model / View / Controller. (Normcore, 2022)

Normcore also offers the possibility to host Normcore on developers' servers or let Normal host a private copy on their cloud infrastructure called Normcore private cloud. The private cloud is maintained 24/7 by Normal DevOps team and can be hosted for example on Amazon Web Services, Google, and Azure. (Normcore, 2022.)

On top of the regular networking framework, Normcore offers also VR solutions such as avatars, which is a digital model of a player-controlled character, and voice chat to quickly prototype VR game concepts. On their documentation, there are simple quick guides to setting up premade avatar prefab or building a custom avatar including voice chat. (Normcore, 2022.)

2.5 Multiplayer solution selection for performance evaluation

This thesis was commissioned by Ade Oy and based on discussion with stakeholder the networking solutions for comparison were selected to be an open-source solution and software as a service solution provider. As these could give more variety for comparison.

For the open-source solution, the selection was between Mirror and Fish-Net as both frameworks are similar according to documentation. Both frameworks are server authoritative utilizing the client-server model and offer an easy set up multiplayer components. Mirror has a solid community behind it and support for Unity LTS releases. Also, lots of feasible tutorials are available to help create

multiplayer with Mirror. When the thesis project was started the Fish-Net multiplayer solution was relatively new and did not have a solid community behind it or many tutorials available. Same as Mirror Fish-Net also provides support for LTS releases. Although, both frameworks offer similar features and Fish-Net is a newer competitor the selected framework was Mirror as for the time being it still seems to be a more solid option.

Software as a service selection was between Normcore and Photon. Normcore seems like a solid option for multiplayer prototyping as it offers premade player avatars and seems to be easy to set up and offers a similar cloud server solution as Photon. There were not many literature or user experience reviews available for Normcore. Photon is a more known multiplayer provider with long history and community behind it. Photons' new multiplayer solution Fusion offers a similar host-client option besides the could server which is natively available with Mirror. Also, according to documentation Fusion is created as a combination of PUN and Bolt and should have better performance. Therefore Photon Fusion was selected for evaluation with Mirror as it would be possible to create similar client-server solutions with both frameworks for a more similar evaluation comparison.

Unfortunately while setting up the Photon Fusion project it turned out to be incompatible with the VR plugin used in the Unity project and had to be replaced with Photon PUN2. This issue is discussed in chapter 4.

2.6 Related literature

Besides the networking tools analysis research report provided by Unity to help developers decide on a proper framework for their game, the available research focuses more on the concept related to performance evaluation in different types of games (Unity, 2021). Also, work studying network performance was available. Here are presented a few related research literature that has helped to design the performance evaluation comparison tests in this thesis research.

Hanse, Jurgens, Makaroff, and Callele in their study discussed performance measurement requirements for creating a performance measurement framework for real-time multiplayer mobile games. The key elements in their study were to monitor and measure bandwidth usage and latency on the network side. Besides monitoring the network, the authors focused on the user's performance indicator which was to monitor the frame rate. (Hanse et al. 2013.)

Lindblom in his research investigated networking performance in VR applications using Mirror library, Unity game engine, and Oculus Quest hmd. Lindblom's study focused on measuring performance by using-client server architecture on determining the performance of VR applications when user counts increase on host runs in Oculus Quest. Lindblom's study measured frame rates with different user counts and server sync intervals. Lindblom's study revealed 18 players as the maximum number that the host on the Oculus Quest device was able to handle without frame rate dropping under 72 fps. (Linblom, 2020.)

Jinjia and Dongliang discuss in their paper the architecture of VR video streaming. The article mentions a challenge being on transferring high-quality panoramic VR video. The main challenges are in compensating for the gap between user experience and limited network capacity. The challenges were divided into network computing power, communication efficiency, and network service latency. (Jinjia & Dongliang, 2021.)

Umeh, Akpado, Okechukwu, and Ejiofor discuss in their paper network monitoring tools to measure throughput and delay performance. Their study includes network measurement tools such as Netstress, Wireshark, and Jperf to measure throughput, bandwidth, latency packet sizes, and traffic with TCP and UDP protocols. (Umeh et al. 2015.)

3 Networking architecture for multiplayer games

To better understand the purpose of these multiplayer frameworks and services, mentioned in the second chapter, it is needed first to take a look at multiplayer networking architecture. Architecture describes how the multiplayer game is being built, what components to include, and the topology to use. (Sloan & Khagendra, 2022.) This chapter covers some of the key concepts of multiplayer game architecture.

There are multiple different ways a multiplayer architecture can be created. The game can be played offline as a local multiplayer on a single device or multiple devices through a local area network or the internet. Multiplayer games can use several different networking topologies such as peer-2-peer and client-hosted models or a game can run on a dedicated game server. In the end, the architectural design comes down to what type of game is being made.

3.1 Network topologies

When creating multiplayer games, it is necessary to plan how to send data from one device to another. This is what network topologies are for. Network topology defines the arrangement of all the devices on a network. (Engelbrecht, 2022.) This determines how clients and hosts, and physical or virtual machines are related to one another (Unity Multiplayer Networking, 2022a).

These vary from local multiplayer known as couch multiplayer to networked multiplayer architectures which can be utilized through local area networks or over the internet. Each architecture has its ups and downs and knowing which one to use is determined mostly by the type of game or application that is being developed. When choosing architecture for a VR game it is highly recommended to take into consideration that the connection is stable and provides a smooth lag and jitter-free experience for the player.

3.1.1 LAN

Local area network (LAN) is a term used to describe computers that are connected within a small area. (Glazer & Madhav, 2015.) These can be places like schools, and businesses. Most people with internet connections use LAN networking in their own homes. A basic LAN setup includes a router that directs data traffic between each device internally or over the internet. (Sloan & Khagendra, 2022.)

LAN server starts a server on the user's computer where other users can then join. As LAN is local internet it is possible to connect to any port of any computer internally because routers have minimal restrictions which allow a user to play their games. To join a LAN party players are required to make connections from the same network to access the game session as the router prevents any access from outside the local area network. (Sloan & Khagendra, 2022.)

Benefits for LAN party games are zero latency as all players are connected to the same local network, costs are lower no backend solution is needed, and scalability is better as it is possible to make larger games compared to local multiplayer. The downside still is that player amount is locked to players who are in the same location.

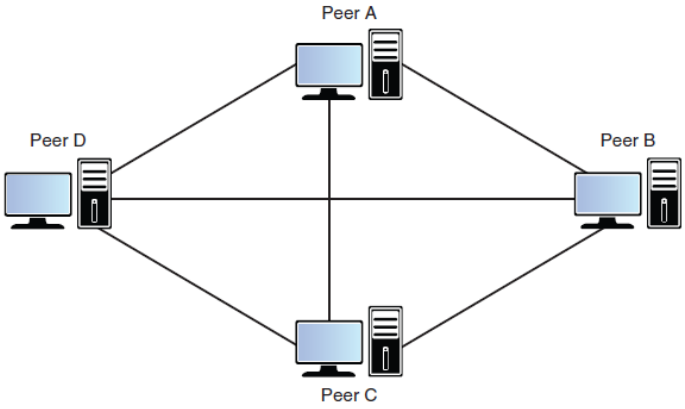
There are ways to get around the location limitation. One way would be to create an open server and open the required port so that anyone from the internet can access it, but this is not very feasible as it requires a static IP and also exposes networking devices and data traffic. This type of solution would require setting up a firewall to have protection from unwanted activity. Another possibility is a VPN tunnel. Network VPN tunnels can allow only trusted parties to connect to it. This way networking devices and data would not be openly available to everyone. (Sloan & Khagendra, 2022.)

Usually, LAN parties are set up by using an external server to players to connect and play the game. For its low latency LAN topology can be beneficial

for locally operated VR applications where the player experience is drastically determined by the visible lag.

3.1.2 Peer-to-Peer

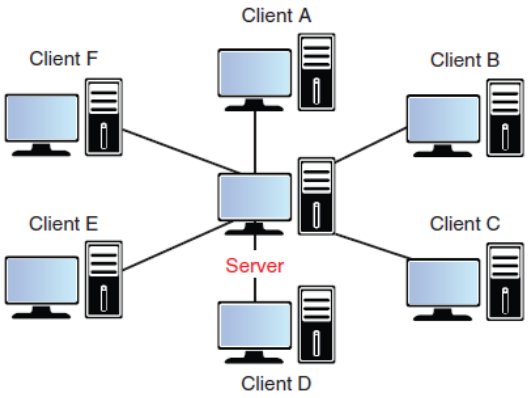
A Peer-to-peer (P2P) network means that multiple devices are connected directly to each other without any entity between them. When talking about Peer-to-Peer networks there are two concepts of P2P. First is direct P2P which uses a mesh topology (Picture 3) where all the players connect to one another to form the multiplayer network and handle the client data synchronization themselves. (Sloan & Khagendra, 2022.) Even though direct P2P is cheap to make there are issues with scalability and security.



Picture 3. Peer-to-peer model. (Glazer & Madhav, 2015)

3.1.3 Client-Server model

The second P2P is hosted client-server model (Picture 4), which is the most used P2P model. In the client-server model, players connect to the server which is responsible for the game's data flow. The server can be either separate or one player is acting as both a server and a client for the game session. (Sloan & Khagendra, 2022.)



Picture 4. Client-Server model. (Glazer & Madhav, 2015)

There are multiple benefits of a P2P network such as no need for separate servers which lowers the costs, no queue management is needed making no need to manage multiple rooms or matchmaking as this is done by the players themselves. There is not much downtime as the players manage most servers, therefore, there is only little dependency on the internet. (Sloan & Khagendra, 2022.)

Downsides are a host advantage as the game session is running on their computers. The further away or due to possible network issues players might get more latency while the host has none. It is also possible for the host to quit before the game has ended which can cause the game session to end or to have some kind of host migration which can also cause delay and ruin the experience. Lastly, there is always the possibility of cheating as it is nearly impossible to prevent the host from cheating when the game is running on their computer.

3.1.4 Dedicated Game Server

Dedicated servers are servers that have been dedicated to serving one purpose. In multiplayer games, this is to run the server build of the game where players around the world can join in. Dedicated servers are preferred because of their power and flexibility. Servers are designed to run around the clock on

each day of the week without issues maintaining the performance. (Sloan & Khagendra, 2022.)

Dedicated servers can be hosted by the company by purchasing the required hardware and storage server room space to place the servers. Besides investing in its own set of dedicated servers there are available other solutions like cloud servers. (Sloan & Khagendra, 2022.)

An option for cloud servers is a software as a service where the seller is renting hardware in a cloud. When hosting a scalable multiplayer game on a global scale cloud hosting can be a good choice. Cloud hosting allows games to be scaled into different regions which can reduce latency and improve the player experience. Large cloud hosts usually allow adding new servers when games player amounts increase and vice versa when player amounts decrease servers can be decommissioned to match the current need of the game. (Engelbrecht, 2022.)

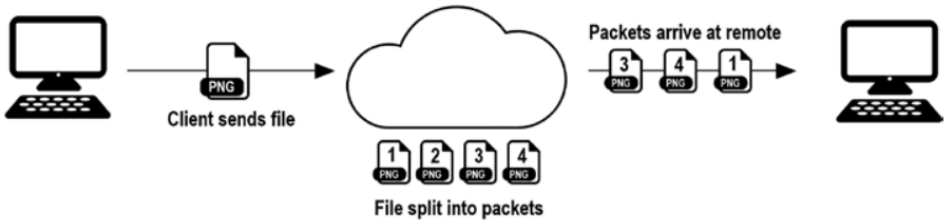
Known cloud server providers are Microsoft Azure Playfab, Google Cloud, and Amazon Web Services. These providers offer different solution plans to choose from making scaling very easy according to games user base growth. Photon and Normcore also offer cloud server capacity on top of their networking solutions.

The benefits of dedicated servers are in performance where it can be chosen how powerful server hardware is and also how many servers a game needs to have to give a good experience for the player. Servers can be located in different areas making it easier to scale up or down and also to decreasing possible latency and so increase performance and player experience. Besides performance, compared to other solutions dedicated servers can provide better security as it is the server who is in charge of the game session and not an individual player. The downside is that all this comes with a cost as acquiring hardware or purchasing cloud services can be quite costly depending on the requirements and user amounts of the game.

3.2 Networking transport layers

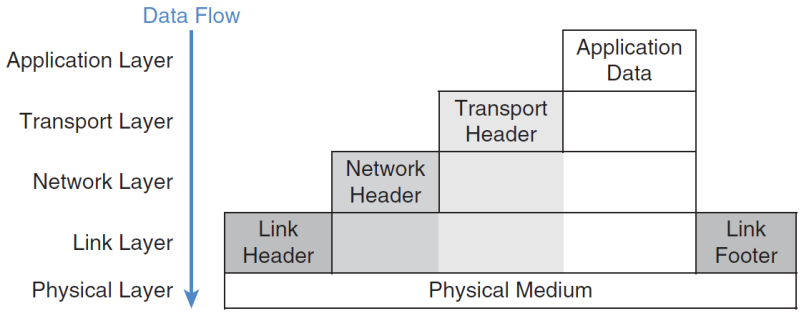
A network is a group of computers communicating with each other by using a shared communication protocol. Two primary standard transport protocols are Transport Control Protocol (TCP) and User Datagram Protocol (UDP). Transport protocols send packets of data between one another to keep the game synchronized. When player 1 moves, then a data packet is sent representing the made movement. Player 2 receives the data package and interprets it and presents the movement of the Player. Multiplayer games use typically either TCP or UDP or a combination of both. TCP and UDP have their differences and it is required to determine which one suits best the game's requirements. (Engelbrecht, 2022.)

Messages sent from one player to another are split into small packets (Picture 5) which are routed through the network. This is done because the packets can be lost in transit or delayed and delivered later. Packets can also arrive out of order. It is for the developer to decide if packet loss is acceptable or not. (Sloan & Khagendra, 2022.)



Picture 5. The sent file is split into smaller packages and received out of order. (Sloan & Khagendra, 2022)

Multiplayer game networking is usually following a Transmission Control Protocol / Internet Protocol (TCP/IP) model which includes five protocol layers (Picture 6). Each layer has its purpose which includes accepting, packaging, forwarding, receiving, and unpacking the game data. (Glazer & Madhav, 2015.)



Picture 6. Data flow. (Glazer & Madhav, 2015)

While working with multiplayer frameworks application layer is the end-user layer where the multiplayer code lives. This layer is responsible for creating and interpreting the data. This can be an example of the damage done to the player or some other type of data that is presented to the player. (Engelbrecht, 2022.)

The second layer is the transport layer which has an important part in designing the multiplayer game architecture. The transport layer is responsible for host-to-host communication and data conversion either to TCP or UDP segments. (Glazer & Madhav, 2015.)

The network layer converts segments into packets and sends them to the network adapter (Engelbrecht, 2022.). Network layer's job is to provide a logical address to the link layer. The most commonly used protocol for this purpose is IPv4 and now the newer IPv6. (Glazer & Madhav, 2015.)

The link-layer provides a method for communication for physically connected hosts. This is when the sent packages are converted into frames which is the single unit of transmission in the link layer. (Glazer & Madhav, 2015.) Once data reaches the physical layer the frames are then transmitted between devices over the network as bits which is the data unit for the physical layer. (Engelbrecht, 2022.)

The most valuable layers to understand for game developers are application and transport layers. (Engelbrecht, 2022.) This is because multiplayer game programmers are mostly working with these layers. Still, to make the game

functional it is necessary to understand other layers and how they affect the layer above them. (Glazer & Madhav, 2015.)

3.2.1 Transport Control Protocol

Transport Control Protocol (TCP) is known as a connection-oriented service. This means that the TCP protocol guarantees that packets are not lost, and messages arrive intact and in order. In a connection-oriented service, the client must have a connection established with a server to ensure the data is sent and received. (Sloan & Khagendra, 2022.) For multiplayer applications, this can cause unwanted latency and is more suitable for slow pasted games as received acknowledgment is required before the next packet is handled.

3.2.2 User Datagram Protocol

User Datagram Protocol (UDP) is known as a connectionless-oriented service. In this protocol, the client is not required to be connected to the server instead it only sends out the information. This will increase the speed as the packets are not required to be delivered in order and packet loss can happen. UDP protocol is suitable for faster pasted multiplayer games, audio, chat, and video streaming. (Sloan & Khagendra, 2022.)

3.3 Data synchronization

An essential part of any multiplayer game is data synchronization. Without synchronized data, a multiplayer game can not function as the actions players are performing are only occurring locally on each player game instance. Different frameworks have their methods for synchronizing data, like variables when they have been updated, to clients. Example Mirror has SyncVar which are variables synchronized from server to clients when the variable value is changed. If something else than the object state needs to be transmitted from client to client RPCs can be used. (Glazer & Madhav, 2015.)

3.3.1 Authority and ownership

Authority in multiplayer games is a way of deciding who has control over a game object. Meaning who is the source of truth when it comes to synchronizing specific data. The data can be movent synchronization, different data values which are presented to all players such as health, score, and even calling specific methods on all client instances. The is server authority where the server has control over an object, which is usually the default owner. This is where the networking tool is built to be server authoritative. Then there is client authority where the client has ownership over a game object. In a direct P2P type of multiplayer example Photon PUN, the master client is holding the ownership of all the game objects and the authority can be transferred to other player instances to keep the game synchronization intact with all connected clients. If a player instance that does not have the ownership over a game object is trying to interact with it the information of the player's actions does not get sent to other player instances and is only occurring locally on this one instance. (Glazer & Madhav, 2015.)

3.3.2 Callbacks

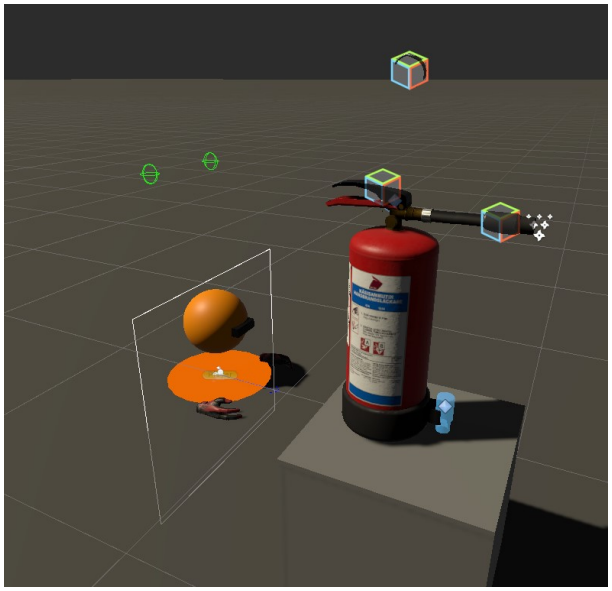
Callback is a function to call another function. These are used in multiplayer frameworks to handle logic when certain events occur. These can be divided into server callbacks and client callbacks. The most usual methods in any multiplayer game are for example related to what happens when the game server is started or stopped, the player joins or leaves a game, or even when the authority over an object is changed from one player to another. Usually, these methods can be overridden to include game-specific custom logic. (Sloan & Khagendra, 2022.)

3.3.3 Remote procedure calls

RPCs are actions players are doing in the game which is causing a procedure to be executed in one or more other clients. These can be sounds, visual effects, and other functions as well. RPCs are usually called on client instances to let the server know that something should happen on other client instances. The server then sends this information to a specific client or all clients. (Glazer & Madhav, 2015.)

4 Multiplayer VR training application development

This thesis was commissioned by Ade Oy, a company that is specialized in virtual reality training application services. The main objective for this thesis project was to implement the multiplayer to Unity project (Picture 7) provided by Ade by using two different multiplayer frameworks and evaluating how they perform.



Picture 7. Example of the base project.

The stakeholder from Ade Oy was Sami Laukkanen and project requirements were planned according to Sami's wishes and the authors' suggestions. It was agreed right at the beginning that, due to the short time available for development, the project would not include a whole usable application. Instead, the project was created as a technical demo which was focusing on measuring and comparing multiplayer frameworks' performance using Unity game engine LTS version 2020.3.

The used project template was a very simple fire extinguisher training game, only containing a scene with a VR player game object, plain as ground, and VR usable fire extinguisher. The VR toolkit that Ade uses in this project template is a custom combination of Steam VR and Tilia VR plugins for Unity. Tilia is also

known as VRTK. Tilia VR plugin is not provided directly for download in Unity's Packet manager, instead, all available components which Tilia provides are listed on their website, and developers can choose the components to be added via Packet manager using a specific download link from the website. Steam VR is a VR tool plugin for Unity provided by Valve. Steam VR offers tools and components to set up VR interactions in a game which is also supported for multiple different devices like HTC Vive, Valve Index, and Rift S.

4.1 Requirements for multiplayer VR application

After going through available multiplayer frameworks with Sami Laukkanen it was decided that the project would be created using an open-source solution Mirror and multiplayer as a service Photon. Mirror was selected as it is a well-known framework with a steady new release schedule and Unity LTS support. It also has a great community and supports even though not many VR-related tutorials or guides are currently available for Mirror. Photon was selected as it is also well known and has a long history of providing Unity multiplayer services. From Photon's available services the Photon Fusion was the first selection for comparison as it is created to substitute PUN and Bolt in the future and includes features from both of these services.

The multiplayer application was first decided to be utilizing host-client architecture which is natively available for Mirror. According to Photon Fusion documentation, this solution would have had a similar hots-client architecture available as Mirror. Unfortunately while testing the Photon Fusion it was discovered that both Fusion and Tilia VR plugins used in Ade's Unity project are using different versions of the Malimbe packet and therefore were incompatible and could not be used. This was an unfortunate setback, but it was then agreed with Sami Laukkanen that Photon PUN2 would be used instead as the author already had some prior experience with it. This had an impact on the project design as the client-server model is not possible with PUN2 and it is also an older, less performant multiplayer solution according to Photon documentation.

This means the server-side performance with host-client architecture can not be evaluated, instead, the evaluation focuses solely on client-side performance.

The multiplayer application was required to have a simple start-up scene acting as a lobby for players to join the session. The start-up scene contains a connection option in world space for VR users and also an option to join the sessions by button on the desktop UI. From the start-up scene, the main game scene is loaded which includes VR Player, two extinguishers, and a few interactable objects synced over the network.

As the time for development was short full-body tracking was decided to be left out of this research. To keep the projects simplistic and executable within the time frame it was decided that VR players would require to have an avatar, which consists of a head and hands whose movements are synced through the network for other players using transform synchronization components available in both frameworks. This means that each player avatar has three child objects which transform is synchronized over the internet. With full-body tracking the amount of tracked objects will increase and as speculative could require custom transform tracking components or a more optimized third-party solution.

The fire extinguisher was required to have synced movement for the base, nozzle, and pin and functionalities triggered by the player controller input when using the extinguisher. The functionalities included for example activating the extinguisher by removing the lock pin and using the fire extinguisher with the action button. An additional requirement was that the player would be able to give the network synced objects to another player from hand to a hand.

4.2 Challenges

The main challenge in creating a comparable project with selected multiplayer frameworks is that Mirror and Photon PUN2 slightly differ from one another where it is possible to client act as a host and server in a local network with Mirror this is not possible with PUN2 as it is a direct P2P solution meaning that each client synchronizes data from everyone else an there is not a single host

but a master client. PUN2 also relies on Photons could server services or Photon’s own separate server solution which can be run on a separate machine.

Challenges arise as well with the VR player provided in the example project as it can not directly be used as network spawned player object because some components used in VR player include singleton scripts which restrict the game scene so that there can be only one Tilia VR player presented locally and only the avatar graphics are synced over the network. Other challenges were on solving how to manage authority change owner game objects so that they are correctly synced for all players simultaneously and, also making it possible to give objects directly from hand to a hand.

It was soon discovered that the project’s hose component used in the extinguisher asset had a bug in it. This made it so that if the scene included multiple extinguishers all hoses were synced according to the extinguisher which was currently moving. To solve this issue the hose component was replaced as a workaround for this study by a script found on Github provided by Mathias Soeholm which renders a cylindrical mesh similarly to Unity’s LineRenderer. (Soeholm, M. 2022)

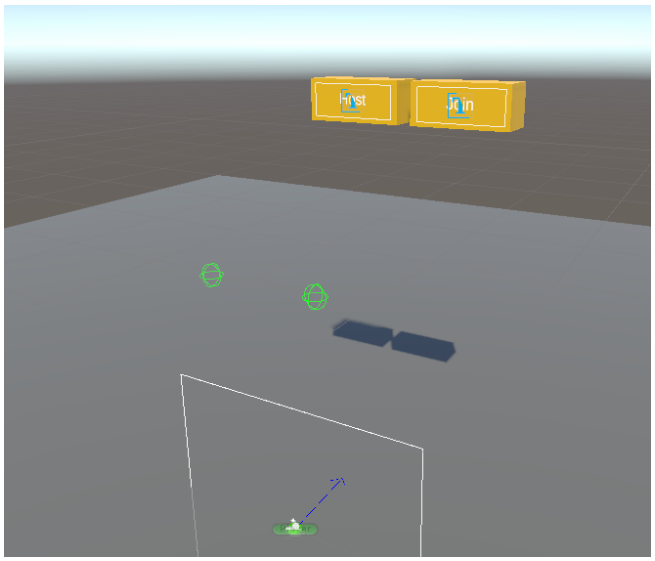
5 Implementing multiplayer solutions

Mirror and Photon PUN2 were selected for performance evaluation comparison. Both frameworks have their own application programming interfaces (API) and ways to implement multiplayer functionalities to VR applications. This chapter goes through the main parts of how both multiplayer solutions were implemented into the Unity project. As a result, two versions for both frameworks were created. One utilized a cloud server to host a game session and the other utilized a local server running on a separate computer.

5.1 Lobby and game scene

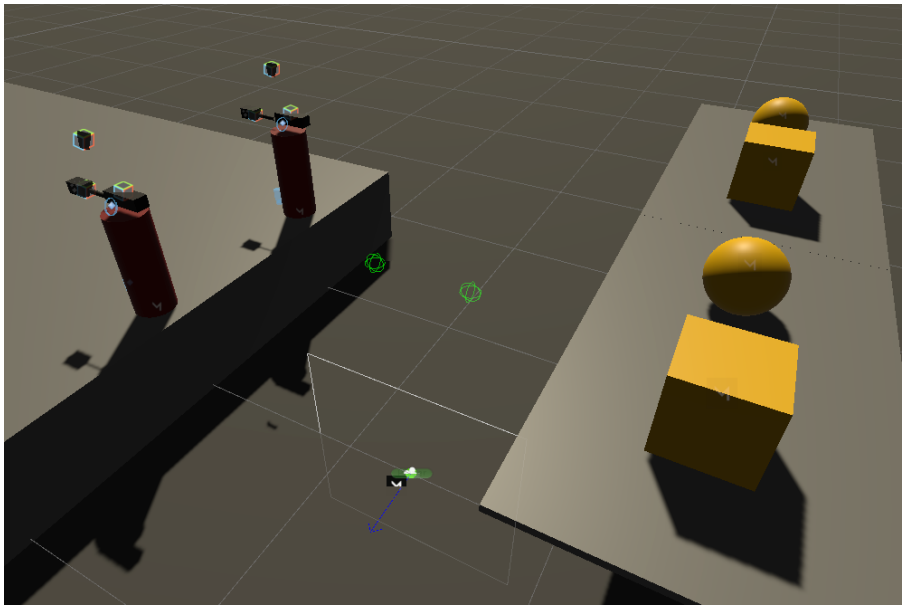
The project build contained a lobby scene (Picture 8) with the possibility to start a new game or join an existing one. Besides having only an on-screen button to press a world space interactable block button was added for VR players to interact with. The onscreen button was placed in the top left corner.

Mirror provides Network Manager HUD component to easily add the Host and joint session buttons. Photon instead provides a script template on their website which was utilized to add the connect to game button on-screen UI.



Picture 8. Example lobby scene from Mirror project with VR buttons.

The game scene (Picture 9) contained a local VR player, two fire extinguishers that are synchronized owner the network, and four other interactable objects to test the object's authority transfer between players. To represent the player location and movement between clients a network synchronized player avatar model is spawned to the game scene when a player joins the game session.



Picture 9. Example game scene from Mirror project.

5.2 Data synchronization

The thesis project was decided to be kept rather minimal for network feature-wise. Therefore, only Mirror's and Photons' own components were used to synchronize the movement for game objects instead of creating custom logic for optimization purposes. Besides each framework's components, all other data synchronization was agreed to be done by using RPCs as both Mirror and Photon provide a similar option for it.

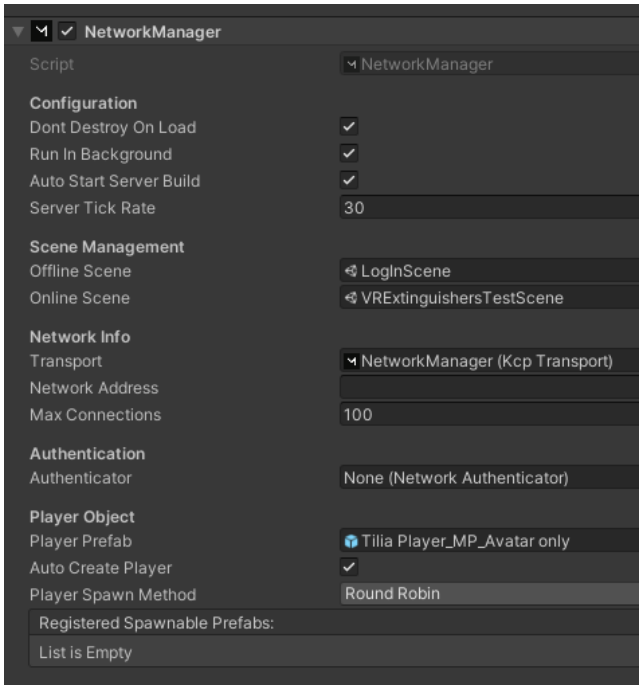
To make game objects networked objects they have to have a network component attached to them. For this purpose, Mirror provides a Network Identity component and the Photons equivalent is the Photon View component. Once the game launches a network ID is provided to each networked object to keep track of them.

It is not enough to have game objects' movement synchronized over the network by adding only the Network Identity or Photon View components to them. In addition, for Mirror a Network Transform component is required for each game object to enable transform synchronization over the network. Mirror also provides a Network Child transform component which can be used to synchronize child objects' movement separately. Mirror requires that all transform components have to be placed on a root game object and the reference to the game object is added to the component.

For Photon PUN2 a Photon Transform View component has to be added to a game object to synchronize the movement of the object. These can be placed directly either to the root game object or a child object. Photon View automatically keeps track of all the game objects with Photon Transform View components.

5.3 Network manager

Network manager is responsible for either hosting a server or establishing the connection to the server. A network manager can be customized to hold a game-specific logic and configurable settings. Mirror provides build-in components for multiple different functionalities and the Network Manager component is one of them. Mirror's default Network manager (Picture 10) works well as a starting point but it is recommended to create a custom Network Manager which inherits this class and adding own game-specific logic to it.



Picture 10. Mirror NetworkManager component.

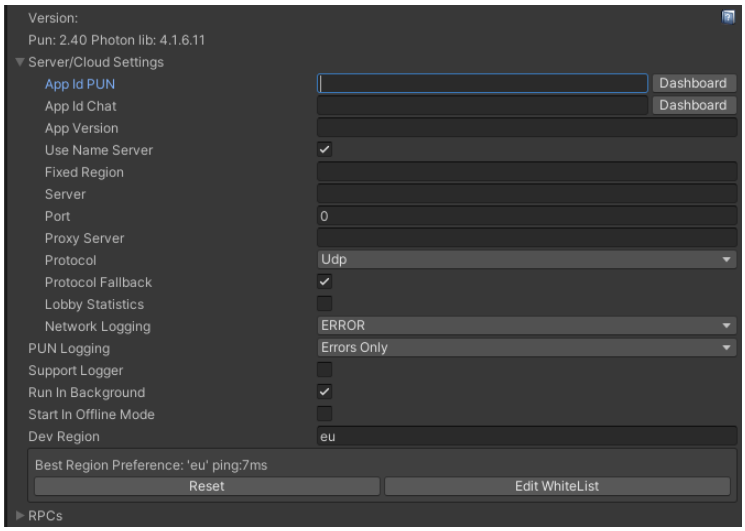
Photon, on the other hand, does not provide a built-in component to just drag and drop into the hierarchy as Mirror does, but instead, developers have to create their own Network Manager scripts using the PhotonNetwork class library. For this thesis project, a simple Network manager-script was created for players to create a game room on to Photon server and for others to join it.

```
public class NetworkManager : MonoBehaviourPunCallbacks
{

            void Start()
    {
                        PhotonNetwork.AutomaticallySyncScene = true;

                        if (!PhotonNetwork.IsConnectedAndReady)
                        {
                                    ConnetToServer();

                        }
                        else
                                    PhotonNetwork.JoinLobby();

    }

            private void ConnetToServer()
            {
                        PhotonNetwork.NickName = $"Player {Random.Range(1,
                        100000)}";
```

```
                PhotonNetwork.ConnectUsingSettings();
                Debug.Log("Connecting to server");
        }

        public void CreateRoom()
        {
                RoomOptions roomOptions = new RoomOptions();
                roomOptions.MaxPlayers = 100;
                roomOptions.IsVisible = true;
                roomOptions.IsOpen = true;

                PhotonNetwork.JoinOrCreateRoom("Default room",
                roomOptions, TypedLobby.Default);
        }

        public void LeaveRoom()
        {
                PhotonNetwork.LeaveRoom();
        }

        #region Photon Callback Methods

        public override void OnConnectedToMaster()
        {
                Debug.Log("Connected to Master Server");

                PhotonNetwork.JoinLobby();
        }

        public override void OnJoinedRoom()
        {
                Debug.Log("Joined room");

        PhotonNetwork.LoadLevel("VRExtinguishersTestScene_Testing");
        }

        public override void OnJoinRoomFailed(short returnCode, string
        message)
        {
                Debug.Log("Joining room failed: " + message);
                PhotonNetwork.JoinLobby();
        }


        public override void OnPlayerEnteredRoom(Player newPlayer)
        {
                Debug.Log("New player joined room");
                base.OnPlayerEnteredRoom(newPlayer);
        }

        #endregion
}
```

To make a Network manager for PUN2, the script must inherit MonoBehaviourPunCallbacks class. Script created for this project first checks if the client is already connected to the server and if not the ConnetToServer method is called otherwise the player is directed to the lobby. To connect

players to the server, a method ConnectToServer is used to establish a connection by using settings in game builds Photon server settings (Picture 11).



Picture 11. Photon server settings.

These settings are used to determine if a Photon cloud server or other local server is used, which transfer protocol to use, and the region to where a connection should be made.

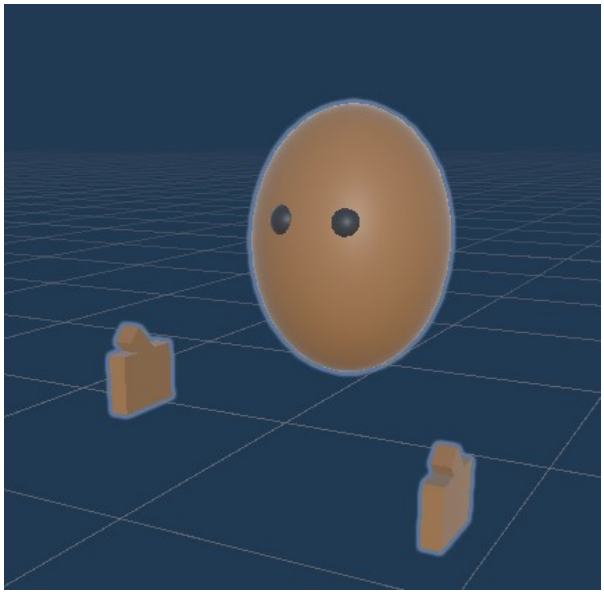
Once the player is connected to the master server the player can either create a room which for this thesis was hardcoded to be named as “Default room”, as it was decided that the project was limited to only this one room for testing purposes.

5.4 VR Player

The main challenge of using Tilia VR and Steam VR plugins in VR multiplayer comes from that some of these components are using singletons. This requires that the VR Player (Picture 12) with the logic components can only exist as a single instance inside the game scene. This made it so that the Tilia VR player object could not be used as a networked game object. The original design was to use one network spawnable VR player where graphics and input logic were separated into two child objects and the game object holding the input logic

would be either enabled or disabled if the player instance is the owner of the player game object or not. To solve this issue, it was decided to only have player visuals synchronized over the network, and YouTuber Valem's tutorial (Valem, 2020.) for creating a multiplayer game for Photon was used as a base for creating Networked Tilia VR Player for both Mirror and Photon PUN2 projects. Instead of the original design where VR player graphics and input logic were separated to

The same solution was used for both Mirror and Photon PUN2 projects to keep both projects inconsistent with each other. In this solution, the networked avatar is spawned into the game scene once the game launches, and the Tilia player which the player is controlling already exists in the scene on launch. Two scripts were used to synchronize VR networked avatar head and hand movement according to the Tilia VR player's position data and connect the local Tilia VR player to the networked player avatar so that the local controller inputs could be utilized to call network synchronized methods of grabbable networked objects.



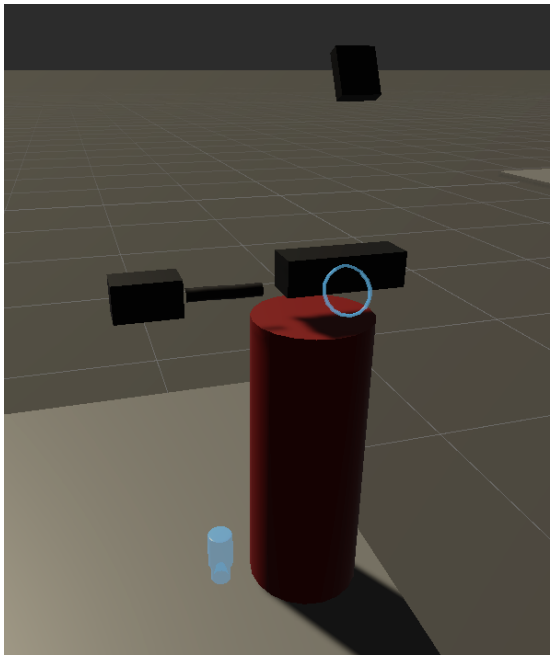
Picture 12. VR player avatar model.

The first script called a SynchronizePlayerAvatar was placed on the player avatar. This script gets the position data from the second script called NetworkAvatarHelper and gives the NetworkIdentity reference to the local Tilia

VR Player. NetworkAvatarHelper script was placed on the local Tilia VR player to form a bridge over to the network synchronized player avatar. This method was used in this project to make it more simplistic to get the reference to NetworkIdentity or PhotonView directly from the local player when the player interacts with objects rather than passing the information from the networked VR Avatar.

5.5 VR interactable fire extinguisher

The fire extinguisher asset (Picture 13) provided in the model project was created from three parts which all were interactable using Tilia's VR Interactions.Interactable component. The extinguisher itself included three interactable main parts, a body, and the handle, a ping to activate the fire extinguisher, and a nozzle. In addition to moving parts, the extinguisher also included two snap zones one for the pin and one for the nozzle.



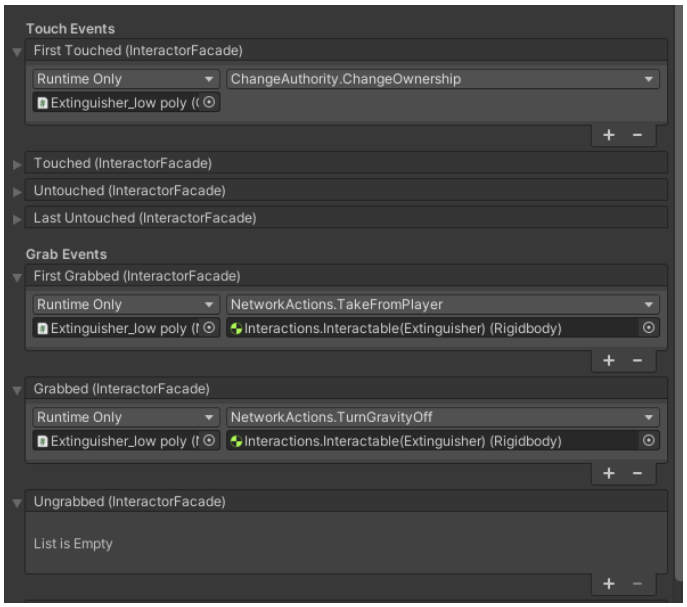
Picture 13. VR fire extinguisher low poly model.

To make the extinguisher a networked object first a network component needed to be added to the root object. The same approach was used as with the VR

player avatar. For Mirror this was the Network Identity and for Photon the Photon view component. Also, to have all the interactable objects network synchronized in Mirror the Network transform child component was used and in Photon the Photon transform view component.

Besides moving parts, the fire extinguisher needed to have different actions to be synchronized as well which are triggered by the local Tilia VR Player's controller inputs. These actions were activating the fire extinguisher's foam by removing the ping from the snap zone and deactivating it by placing it back, activating and deactivating the foam functionality, releasing the nozzle from the snap zone, and turning game objects Rigidbody's gravity on and off. To synchronize these required actions RPCs were used to inform other players that different actions have occurred in the game.

To keep the project simple a single script called NetworkActions was created and placed on the extinguisher object. To link NetworkActions script methods to the extinguisher, the Tilia's Interactions.Interactable components VR Touch Events (Picture 14) were used for this purpose.



Picture 14. Tilia Interactions Touch Events.

The VR Players' hand components included physics colliders, therefore, it was decided to change the authority of the networked object to the player once the player touches an object. Once a player's hand collides with a grabbable objects collider the ChangeOwnership method gets called to transfer the authority of the object over to the current player to synchronize extinguishers movement and networked actions also to other player instances. With the exception, if the other player already has grabbed the object, then authority was not transferred from touch. When giving an object from one player to another the first grabbed event was used to call the TakeFromPlayer method.

The last problem to solve was how to let other players know if the pin or nozzle was removed from the snap zone. For this, the extinguisher has a Tilia Snap Zone Facade component which included zone events that could be used similarly to touch events. Snap zone events include snapped and unsnapped events where the networked methods from the network action script could be called to let other players know the object had been unsnapped. These events could be used also to call other logic like releasing a ping activated the extinguisher's foam functionality.

5.6 Objects authority transfer

Authority owner networked objects are handled slightly differently in Mirror and Photon PUN. Mirror is server authoritative therefore server holds the authority over networked objects by default. Photon, on the other hand, sets the authority to the master client by default which is the owner of the game room. To change the authority or ownership of the game object between players both Mirror and Photon provide a method for this which can be used.

Mirror's Network Identity class has the methods RemoveClientAuthority and AssingClientAuthority which takes the server's connection to the player client and changes the authority owe to that specific player. This can be used as a command method for the server.

```
[Command(requiresAuthority = false)] // Commands can only be sent if the player
has authority to a game object the script is placed unless the required authority
is false

            private void CmdChangeOwnership(NetworkIdentity networkObject)
            {

                        if (networkObject.hasAuthority) { return; } // if player
                        already has authority then return


                        networkObject.RemoveClientAuthority(); // Remove player
                        authority from previous player

                        networkObject.AssignClientAuthority(connectionToClient);
                        // assign authority to new player

            }
```

Where the networkObject is the object to which the player connectionToClient is given authority's owner. If the player does not have authority over the game object neither the positions nor RPCs are synced owner network.

Photon does this a bit easier than Mirror. To change the ownership of a game object Photon has a method in the Photon view class called ReguestOwnership which can be called for the Photon view component attached to a networked object to request ownership over it.

```
public void TransferOwnership()
            {
                        if (_photonView.IsMine) { return; }

                        _photonView.RequestOwnership();

            }
```

This calls automatically Pun ownership callback methods and for this, a separate OwnershipManager-script was created as these callback methods can only have one instance at a time in the game scene. (Photon Engine, 2022.)

```
public class OwnershipManager : MonoBehaviourPunCallbacks, IPunOwnershipCallbacks
{
            public void OnOwnershipRequest(PhotonView targetView, Player
            requestingPlayer)
            {
                        Debug.Log("Reguested: " + targetView.name + " to " +
                        requestingPlayer.NickName);

                        targetView.TransferOwnership(requestingPlayer);

            }
```

```
        public void OnOwnershipTransfered(PhotonView targetView, Player
        previousOwner)
        {
                Debug.Log("Trasferred: " + targetView.name + " from " +
                previousOwner.NickName);
        }

        public void OnOwnershipTransferFailed(PhotonView targetView, Player
        senderOfFailedRequest)
        {
                Debug.Log("Ownership transfer failed to " +
                senderOfFailedRequest);
        }

}
```

The OnOwnershipRequest then calls the Photon views TrasnferOwnership method. Compared to Mirror where the Network identity to which authority to change has to be specified in the code by the developer, PUN does this in the background and what is left for the developer is simply to call the RequestOwnership method.

5.7 Project server design

Because the originally planned host-client comparison research method was not available with Photon PUN2 for testing purposes for this study two client-server versions of the project were created for both frameworks. One for a cloud server and one to act as a headless server on a separate machine. This had a slight impact on the outcome of this study as this agreed approach for separate servers only measures the performance on the client-side whereas in a host-client method it would have been possible to also see the performance impact between these two frameworks for the server while player count increases.

5.7.1 Cloud server solution

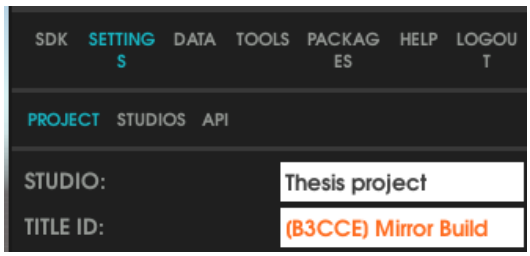
Photon provides natively its cloud server hosting for PUN2. Photons free tier allows only a maximum of 20 player connections for a room. Therefore, this limits drastically the amount of stress that can be addressed to Photon client build and could cause an issue of not being able to have a large enough client

amount running simultaneously to have a proper distinction in the results between the two frameworks.

Adding the photon cloud connection to the Unity project has been made very easy. The user has to have a free Photon user account and create a new PUN2 application from the dashboard on the Photon website. Once a new application is created Photon provides an application ID number which is placed in the project's Photon server settings and the cloud server is already ready to use. (Photon Engine, 2022.)

For Mirror this is not as simple. Mirror is a free framework that does not provide any external server solutions and this part is left solely to developers to decide which approach to take. In this study, it was decided with Sami Laukkanen that Microsoft Azure Playfab would be used as a cloud server solution.

To be able to use Playfabs services a Playfab software development toolkit had to be downloaded from their website and imported to the Unity project. From within the project, the developer must log into the Azure account which can be created for free. After the user account is created similar way as with Photon developer has to create a new studio and game title on the Playfab website. This information is required for Playfab unity project settings (Picture 15) to link the project to a specific title.



Picture 15. Example of Playfab project settings.

After the settings have been added, the user is required to activate multiplayer functionality from the dashboard of the Playfab website. Once multiplayer is activated a virtual server can be created to run a server build of the project. This has to be uploaded to Playfab servers when selecting the specifications of what

kind of virtual machine to use. After the build has been deployed Playfab provides an app ID to connect the Unity project to a specific build.

With Mirror some additional steps are required to make the Playfab server connections work. For this, a guide provided by Angda Lambda (Lambda 2021) was used to create the required scripts to make Mirror builds work with Playfab.

5.7.2 Headless server solution

The free tiers of cloud servers do not always provide the best network performance as the paid subscription always are put first. This might cause some unwanted distortion in measured statistics and therefore does not provide a stable environment for testing as a local area network.

Mirror frameworks can run a client as a server or build just as a server already out of the box. Unity provides the option to create a server build either for Linux or Windows in the build settings which can be run as a headless server. This was used to create a windows server build for this thesis to run separately in the local area network.

Photon PUN2 does not support the same server option as Mirror. Instead, Photon offers a separate Photon Server V5 solution which is a completely standalone server compatible to host PUN2 game sessions. Photon also provides an easy to follow five steps guide to set up the server which was used to get the Photon server running with the thesis project build. (Photon Documentation, 2022.)

6 Evaluation of multiplayer solutions' performance

This chapter consists of the performance evaluation performed for two selected frameworks, the metrics which were chosen, and the results discovered from the comparison.

First tests using the cloud server options with the VR player client revealed that the 20-player limit was not enough to have a proper evaluation of the possible difference in performance. Besides the low player limit for Photon PUN, constant connection losses were experienced with Mirror's Playfab cloud server solution.

Due to these issues and limitations of cloud servers, multiplayer performances were tested in the local area network in Turku FIT center premises using headless servers on a separate machine to have a more stable environment and higher player count than it is possible to get from free tier on cloud server providers. Unity provides the option for server build of the Mirror version with unlimited players and Photon server V5's free license offers up to 100 players maximum limit. For this research, 50 players were decided to be the maximum amount of client instances in one session.

To have more precise information regarding the actual framework's performance, all graphics from the extinguisher were removed and replaced with primitive objects such as cubes and cylinders. This same approach was used with the player avatar. This was decided as during the preliminary tests, it was noticed that the original extinguishers' tris count was rather high, and having multiple objects of this type alone in the scene was causing a noticeable impact on applications performance.

6.1 Research methods

The evaluation was conducted as experimental research. The research design was based on the noisy neighbor anti-pattern where the system shares

resources between clients and the activity of one client could have a negative impact on another client (Microsoft Docs. 2022.).

The research was carried out using a development build instead of Unity editor where network stress was increased in 10-player connection intervals starting from two players and going up to 50 players in total. Each player has three network synchronized body parts. On each player count, a data set was recorded for comparison between frameworks. Besides the measured data, also user experiences relating to noticeable visible changes in performance were observed. The player count was increased up to 50 by using simulated bot clients as not enough VR devices or test users were available.

UDP was decided to be used as a transfer protocol as even if some packages are lost in transit UDP is more suitable for faster-pasted games such as multiple VR users moving simultaneously. In comparison, TCP requires a packet received acknowledgment which can cause some unwanted slowness in the application. Also, keeping in mind that as a further development possibility more features could be added such as Voice over Internet Protocol (VoIP) which is more suitable to be used with UDP than TCP. (Sloan & Khagendra, 2022.) Currently, UDP is the default transfer protocol used with both Mirror and Photon PUN2.

VR client measurements were made using Meta Quest 2 virtual reality headset and computer with the following specs:

Core(TM) i7-8750H CPU 2.20 GHz, RAM 16,0 Gt and Nvidia GeForce RTX 2060 graphics card.

The vsync was turned off in Unity project settings to disable the fps synchronization according to the used displays maximum refresh rate to see the maximum fps for each amount of client instances in the scene. The maximum fps were recorded simultaneously on a desktop client with a VR client on a different computer, this was because Steam VR natively adjusts the fps to match the HMD’s refresh rate. The computer used to capture fps on desktop client had the following specs:

Core(TM) i9-9900 CPU 3.10 GHz, RAM 64,0 Gt and Nvidia GeForce RTX 2080 Ti graphics card.

6.2 Measurement metrics

This research aimed to measure and compare the effects of client instance increase on applications performance, and network traffic, and test user observations of usability on any noticeable visual changes. Measurement was carried out in runtime on the client instance of a game build.

The following metrics were used to measure performance:

- FPS stability,
- CPU usage percentage,
- amount of allocated garbage collection in KB,
- system memory usage in MB,
- latency in milliseconds.

Network traffic was monitored and measured on each player count using Wireshark to capture and compare data from:

- number of packets,
- length of packets,
- used bandwidth.

User experience observation consisted of observing visible performance stability in VR clients such as movement synchronization fluidity and possible visible delay and jitter in the application.

As an honorary mentioning the first test design also included a latency comparison of RPC calls between the two frameworks that were designed, but it turned out to be unusable. As for where Mirror is server authoritative and to call RPC the client first sends a command to the server to run the RPC method, the PUN2 is not server authoritative, and instead, the RPC is run locally on a sending client and the RPC data is delivered via server to other clients. This

was noticed as the latency for Mirror shifted between 14 to 40 milliseconds whereas with PUN2 it took on an average of 0,06 milliseconds while the average round trip time was 10 milliseconds.

6.3 Tools to measure performance

6.3.1 Wireshark

Wireshark is a tool that Photon also suggests to be used in measuring the network traffic. Wireshark states to be the widely-used network protocol analyzer used by many commercial and non-profit enterprises, government agencies, and educational institutions. With Wireshark, it is possible to capture live traffic of specified ports and protocols and store it in to file for later offline analysis of transferred packets. (Wireshark. 2022a)

In this evaluation research, Wireshark was used to capture the packet transfer of UDP protocol and to compare packet lengths and bitrate per second for each player-level test case. To have more accurate results Wireshark can be operated from the command prompt by giving a startup command which can include specific parameters for the amount of time the network transfer is being recorded, the source if it is ethernet or Wifi, the used pre-saved capture filter, and file path and name the date is stored to. (Wireshark. 2022b)

6.3.2 Scripts for collecting data

The research was conducted by using a development build instead of a Unity editor to measure and record data to a CSV file via script for later comparison. For this purpose, scripts were used to calculate latency, FPS, and CPU usage percentage. As the data recorded from the Unity development build was done on runtime, Unity’s ProfilerRecorder API was utilized in a script to read and record profiler data from GC allocation and system memory usage.

6.4 Research results

Results shown on graphs are average values from recoded data on each player count starting from the baseline with a minimum user amount of 2 players to 10, 20, 30, 40, and up to 50 players in total. Data were recorded from both client-side performance and network traffic.

6.4.1 Application client performance

The data presented on graphs are measured on VR client instances on each measurement metric. With the exception where maximum fps comparison data was recorded on a separate computer from a desktop client simultaneously with the VR client in the same game session.

The project used the Tilia Steam VR plugin and Steam VR stabilizes fps automatically to match the connected hmd's refresh rate which in this case was 72 fps as the refresh rate on Quest 2 is 72 Hz. As expected, the fps for both Mirror and PUN2 were stable at 72 fps on each player count (Figure 1).

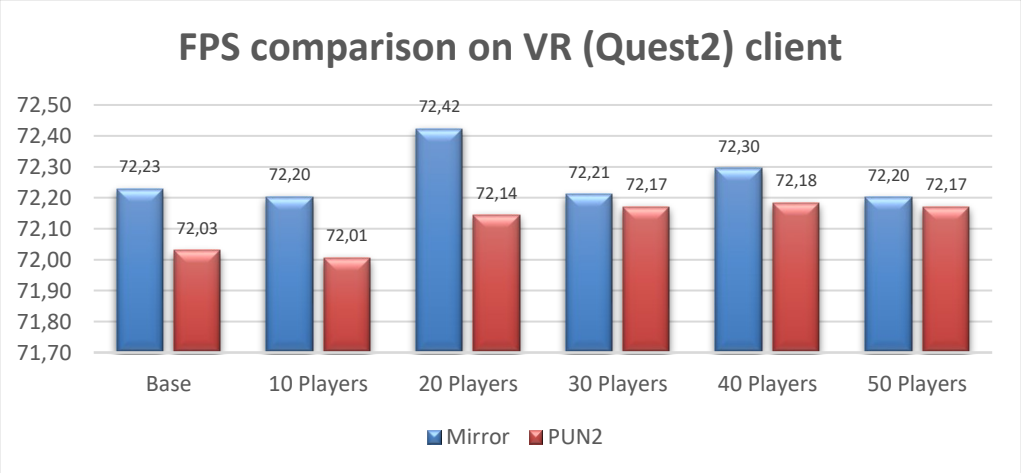


Figure 1. Average FPS comparison on VR client.

The maximum fps on the desktop client (Figure 2) for both frameworks showed that fps dropped significantly more on Mirror. The drop with Mirror was from the baseline with two players on 1372,59 FPS to 50 players with 574,36 FPS making the FPS total drop 798,23 FPS. The drop in FPS with PUN2 was not as great dropping from 1349,42 FPS to 980,95 FPS making the total drop to be only 368,47 FPS. On both frameworks, the fps was quite stable as no large variations were noticed between min and max fps value changes.

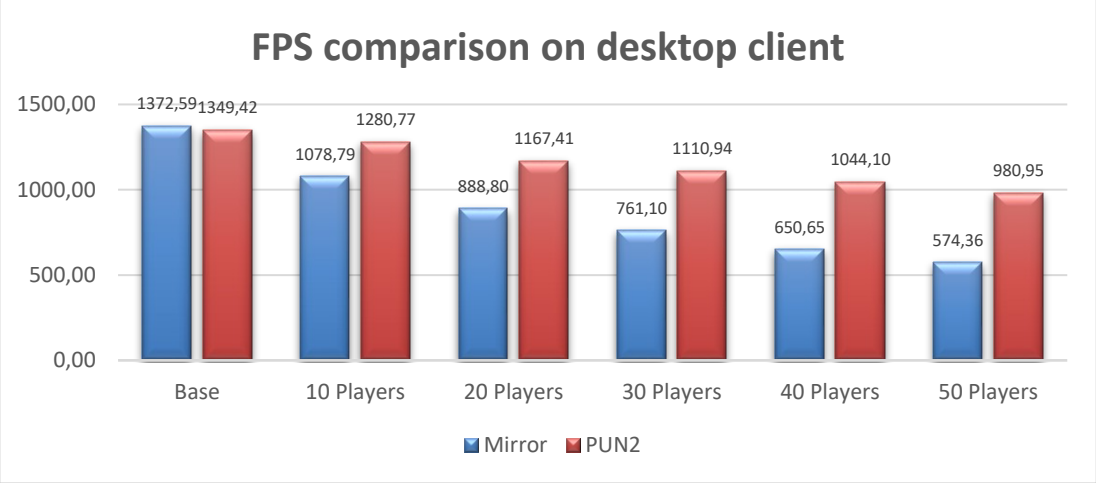


Figure 2. Average FPS comparison on the desktop client.

CPU usage percentages (Figure 3) on both frameworks were similar between 30-40% and no significant differences were noticed in the measured results.

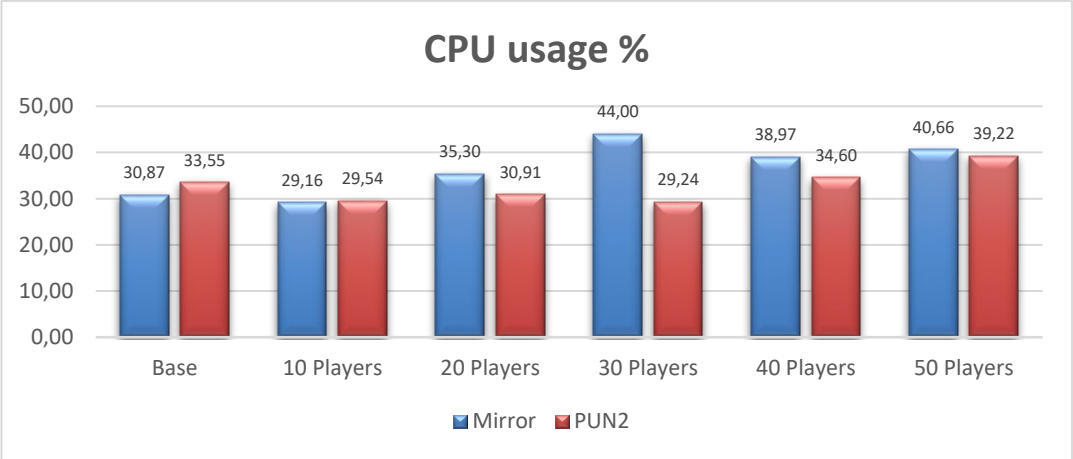


Figure 3. Average CPU usage percentage on the client.

The average amount of allocated GC (Figure 6) between the two frameworks showed surprisingly that Mirror builds GC allocation increased by 10-20 KB with every 10 new players joined to the session whereas PUN2 GC allocation remained more stable between 30-40 KB. Some spikes in GC allocation were also noticed for both frameworks. As the full profiler metrics were not recorded it could not be determined where this difference is coming from and would require more in-depth investigation.

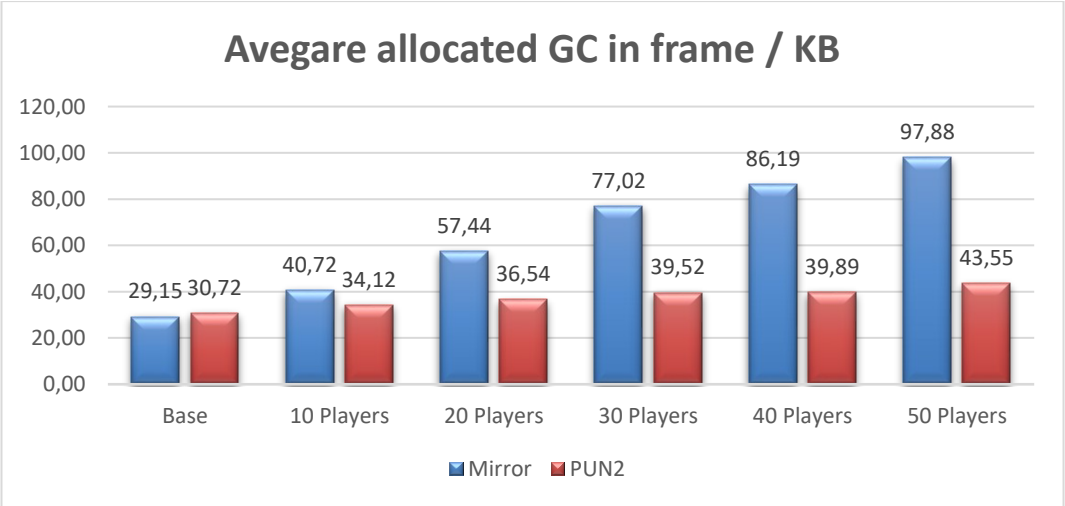


Figure 4. Average allocated garbage collection in the frame.

The average system memory usage (Figure 5) for Mirror build was seen to be more stable compared to PUN2 where some variations were noticed between different player counts.

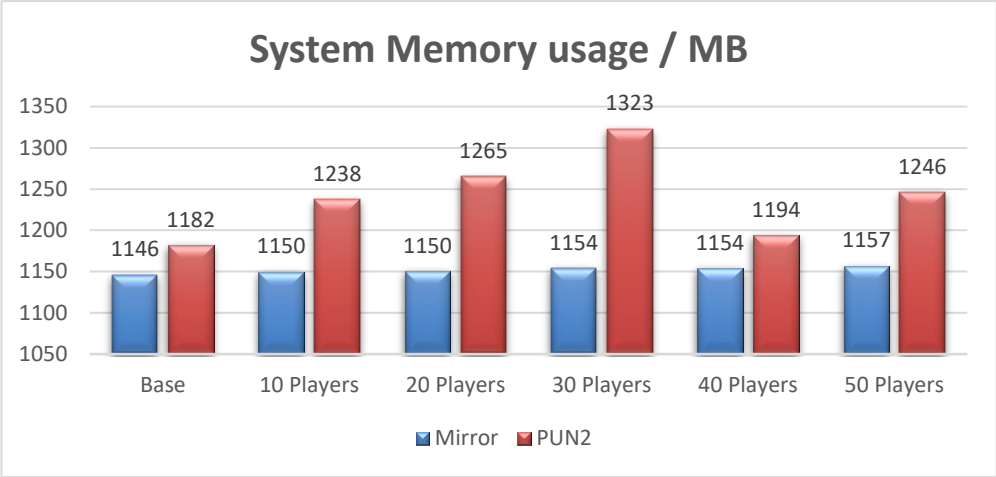


Figure 5. Average system memory usage.

As a disclaimer, the latency between client and server (Figure 6) is not comparable directly one to one as the Mirror build used the Unity's server build whereas PUN2 used the Photon's own more optimized separate server solution. To have more comparable results both frameworks would require to use of similar server solutions. This was a directive comparison that showed both Mirror and PUN2 latency being stable on each player count.

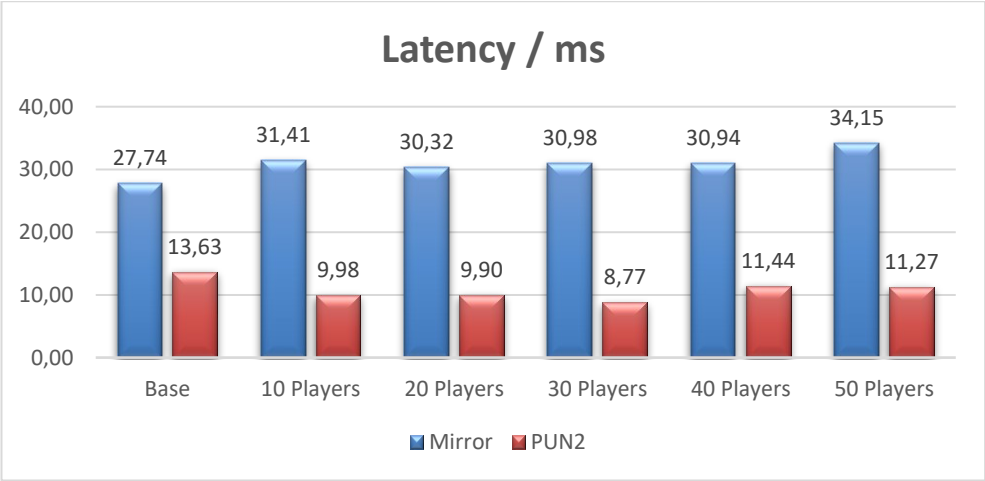


Figure 6. Average latency in milliseconds.

6.4.2 Test users' observations

Test user's observations indicated that Photon PUN2's transform components synchronization might not be as performant as Mirror equivalents. This was seen where fps on PUN2 seemed to overcome Mirror's in measured test data, the noticeable jitter and delay in visual performance of game objects transform synchronization was much more severe with PUN2 compared to Mirror.

On 20 players in the scene, the transform synchronization with PUN2 started to show a more noticeable jitter and delay in movement compared to Mirror where only minor jitter or delay was observed. When player counts increased the PUN2 synchronization delays were getting much worse as when reaching 40-50 players with PUN2 constant full stops in transform synchronization were noticed whereas on Mirror the number of delay increases in transform synchronization was more subtle and no full stops were noticed even on higher player counts. This made Mirror builds end-user usability more stable compared to PUN2 on a higher player count.

6.4.3 Comparison of network traffic

Network traffic measurements (Table 1) show that on Mirror packet count sent through the network in 90 seconds duration is noticeably larger and the average packet sizes smaller than Photon PUN2. The minimum and maximum packet sizes were roughly the same sizes. The larger data traffic on sent packets with Mirror could come with a greater cost as the amount of data sent through the network is higher compared to PUN2. Some contaminated packets were discovered with Mirror whereas none was discovered with Photon PUN2.

	MIRROR				PUN2			
	Packet Count	Average length in bytes	Min	Max	Packet Count	Average length in bytes	Min	Max
BASE	7864	166,44	61	499	2355	266,92	66	611
10 PLAYERS	7125	359,18	61	1241	4860	455,94	66	1238
20 PLAYERS	10034	474,88	61	1241	6468	648,52	66	1231
30 PLAYERS	10167	699,06	61	1241	8194	789,58	66	1234
40 PLAYERS	11901	757,49	61	1241	8247	1005,08	66	1231
50 PLAYERS	13382	797,90	61	1241	9681	1043,74	66	1231

Table 1. Network traffic packet comparison between Mirror and PUN2.

Bandwidth usage (Table 2) showed some differences between these two frameworks. The data collected during testing revealed that PUN2 consumes slightly less bandwidth than Mirror. The stored data were collected in 90 seconds capture intervals, and a longer capture time could reveal the difference much better as more data is transmitted.

	MIRROR	PUN2	
	Bits / s	Bits / s	Difference
BASE	116 k	56 k	60 k
10 PLAYERS	227 k	197 k	30 k
20 PLAYERS	422 k	373 k	49 k
30 PLAYERS	632 k	574 k	58 k
40 PLAYERS	800 k	736 k	64 k
50 PLAYERS	947 k	898 k	49 k

Table 2. The bandwidth of the UDP protocol relative to the capture time.

6.5 Further development suggestions

The performance evaluation in this thesis was executed with very basic multiplayer functionalities in both projects. Besides the project being a technical demo, it was not possible to do the evaluation using the originally selected

frameworks due to incompatibility issues. For future development, the incompatibility issues might get sorted in future releases of the Fusion SDK and Tilia VR plugin therefore it would be suitable to consider redoing the evaluation using this latest solution from Photon. According to Photon, Fusion uses a lot less bandwidth and CPU on the server. This could also indicate that Fusion could have better scalability over Mirror, but this assumption can only be confirmed in one way or another with further studies.

These tests currently only measured the performance on the client-side, therefore for future development, it would be suitable to have also tests that measure the server-side. This could be approached by comparing frameworks that provide the option for the client to also act as a server.

The evaluated projects could have more features included in the future such as VoIP which is a necessary feature in developing VR training applications as the communication channel as traditional chat where the user types the message is not as feasible in VR applications. For example, Photon offers a VoIP service called Photon voice for PUN2. There are also other assets available for VoIP in the Unity asset store like Dissonance audio which is compatible with Mirror and PUN2. Besides VoIP also adding animation synchronization is essential for VR immersion. Both Mirror and PUN2 provide a component to synchronize animations over the internet. Animation synchronization could be useful for example synchronizing players' hand movements for individual fingers. As synchronizing transforms for individual joins adds more network traffic for having to track each joint separately.

Future development could include full-body tracking as for this thesis only players' heads and hands were tracked and movement was synchronized over the internet. One way to include full-body tracking could be to use additional HTC Vive trackers to create a more immersive VR experience (Exyte, 2022). This could be done by adding trackers to players' legs and pelvis area and then animating legs, arms, and pelvis movement by using inverse kinematics. Inverse kinematics moves the joins backward in space which can be useful to mimic the actual human joint movement (Unity documentation. 2022). Having

more tracked child objects in one VR multiplayer avatar increases the amount of networked data and it could be considered to create a more optimized custom transform synchronization or use an available asset such as Smooth synch which is available in the Unity asset store.

This project only consisted of including networking components provided in the framework. It is possible to use the third-party asset or write your own logic for transform synchronization to optimize the performance in this area. Also, only RPCs were used in this thesis to transfer data of action performed by the player. For further development, synchronized variables could also be included to store data and present it for example on a scoreboard. Also, latency compensation and client-side prediction are something that could be included in future studies.

The test performed in this thesis could also be used to evaluate different frameworks' performance in a specific complete application to determine suitability between different available frameworks for different types of applications.

7 Conclusion

The main objective of this thesis was to evaluate the performance of two frameworks in a VR application. The VR application was kept simple in design as the time for development was short and the author had very little prior knowledge of multiplayer game development. The two frameworks selected for evaluation were the open-source solution Mirror and software as a service solution Photon PUN2.

The thesis also covered the core concepts of multiplayer game architecture to better understand the purpose for which these frameworks are being used. The core concepts included different architecture models used in designing multiplayer games.

The thesis project's multiplayer framework implementation encountered a few challenges due to the VR plugin including singleton components preventing the VR avatar to be spawned as a networked object. Instead, the player-controlled avatar had to be instantiated locally without network functionalities and only the player avatar model was synchronized over the internet. The main challenge in this thesis was that the originally planned client-server model comparison, where one client acts as the server was not possible with PUN2. The Photon Fusion was the original choice for comparison but it had compatibility issues with the VR player plugin used in the project and eventually, it could not be used.

The performance evaluation was carried out in the local area network of Turku FIT Center's premises and the data collection focused on changes in application performance, user observation, and network traffic. The data analysis showed that on statistics PUN2 seemed to outperform Mirror on maximum FPS and allocated garbage collection. The performance of CPU usage was similar in both frameworks, but system usage showed more stability in the Mirror framework. On the network side, the sent packet count was higher on Mirror than on PUN2 but the average lengths were smaller on Mirror. Results also showed that PUN2 used slightly less bandwidth than Mirror.

The most noticeable and foremost difference between Mirror and Photon PUN2 came from user observation. Here, it was observed that despite PUN2 outperforming Mirror in some areas on measured data such as holding a higher frame rate with a higher user count than Mirror, the user experience was much worse with PUN2 as more noticeable delays in movement synchronization were experienced in much lower user counts than with Mirror.

This observation indicated that Mirror has better scalability in user experience wise and is a more suitable option with a type of VR application where user counts are 20 or over. It could be possible that Mirror's built-in transform synchronization component performs better compared to the PUN2 equivalent component. PUN2 holds good user experience stability for up to 10 users but starts to show a delay in object transform synchronization when reaching 20 users and above, thus making PUN2 still a good option for smaller VR multiplayer applications. These results were similar to Unity's multiplayer framework survey research regarding the Mirror's and PUN2's performance and scalability.

Lindblom's (2020) study focused on measuring performance utilizing Mirrors client-server model on Meta Quest 2 by measuring frame rate stability while the player count increases reveling optimal threshold to ne 18 players. The goal for this thesis research was similar including also measuring application performance in other aspects as well. The thesis results only showed measured performance on the client-side, but the results from the client-server model would be interesting to see how it differs from the results gathered from client-side measurements and compared to Lindblom's results.

Therefore, in the future Mirror could be compared to some other framework that can utilize the client-server model where one client also acts as a server. With this method, it would be possible to also evaluate the impact of user count increase on the server. It is possible that the incompatibility issues with Photon Fusion will be resolved in future releases and Photon Fusion could be considered again as a promising candidate.

References

Engelbrecht, D. 2022. Building Multiplayer Games in Unity: Using Mirror Networking. New York, USA: Appress. 235 p. ISBN-13 (electronic): 978-1-4842-7474-3. SBN-13 (pbk): 978-1-4842-7473-6.

Exyte. 2022. VR full body tracking: do you really need it. Referenced 14.6.2022 Available at: https://exyte.com/blog/vr-full-body-tracking

Fish-Net. 2022. Fish-Net: Networking Evolved. Referenced 4.2.2022 Available at : https://fish-networking.gitbook.io/docs/

Glazer, J., Madhav, S. 2015. Multiplayer Game Programming: Architecting Networked Games. Boston, USA: Pearson Education. 365 p. ISBN-13: 978-013-403430-0

Hanse, C. et al., 2013. Network Performance Measurement Framework for Real-Time Multiplayer Mobile Games. Conference: 2013 12th Annual Workshop on Network and System Support for Games (NetGames). DOI:10.1109/NetGames.2013.6820601. Referenced 25.4.2022 Available at : https://www.researchgate.net/publication/269305685_Network_performance_measurement_framework_for_real-time_multiplayer_mobile_games

Jinjia, R., Dongliang, X. 2021. Networked VR: State of the Art, Solutions, and Challenges. Referenced 26.4.2022 Available at : https://mdpi-res.com/d_attachment/electronics/electronics-10-00166/article_deploy/electronics-10-00166-v4.pdf

Lambda, A. 2021. PlayFab's Integration with Mirror Unity. Referenced 10.4.2022 Available at: https://angadlamba21.medium.com/playfabs-integration-with-mirror-unity-da998abe2b2a

Lindblom. A. 2020. A Study of Networking Performance in a Multi-user VR Environment Using Unity and the Mirror library. Luleå University of Technology. Referenced 28.4.2022 Available at : https://www.diva-portal.org/smash/get/diva2:1440801/FULLTEXT01.pdf

Microsoft Docs. 2022. Noisy Neighbour antipattern. Referenced 10.4.2022 Available at : https://docs.microsoft.com/en-us/azure/architecture/antipatterns/noisy-neighbor/noisy-neighbor

Mirror. 2022. Mirror Networking. Referenced 4.2.2022 Available at : https://mirror-networking.gitbook.io/docs/

Normcore. 2022. Seamless multiplayer for unity. Referenced 6.2.2022 Available at: https://normcore.io/

Photon Documentation. 2022. Starting Photon in 5 Minutes. Referenced 29.4.2022 Available at : https://doc.photonengine.com/en-us/server/v5/getting-started/photon-server-in-5min

Photon Engine. 2022. We make multiplayer simple. Referenced 6.2.2022 Available at : https://www.photonengine.com/en-US/Photon

Sloan, K., Khagendra, K. 2022. Unity Networking Fundamentals: Creating Multiplayer Games with Unity. New York, USA: Appress. 266 p. ISBN-13 (electronic): 978-1-4842-7358-6. ISBN-13 (pbk): 978-1-4842-7357-9

Soeholm, M. Github. TubeRendere.cs. Referenced 25.3.2022 Available at: https://gist.github.com/mathiassoeholm/15f3eeda606e9be543165360615c8bef

Statzer, J. 2022. VR Expansion Plugin. Referenced 15.6.2022 Available at: https://vreue4.com/

Umeh, O. et al., 2015. Throughput and Delay Analysis in a Real Time Network. International Journal of Engineering and Applied Sciences (IJEAS). ISSN: 2394-3661, Volume-2, Issue-12, December 2015. Referenced 26.4.2022 Available at : https://www.researchgate.net/profile/Godson-Okechukwu/publication/324418934_Throughput_and_Delay_Analysis_in_a_Real_Time_Network/links/5acd4b6ca6fdcc87840a0cd4/Throughput-and-Delay-Analysis-in-a-Real-Time-Network.pdf

Unity. 2021. Choosing the right netcode for your Unity multiplayer game. Referenced 16.5.2022 Available at : https://images.response.unity3d.com/Web/Unity/%7B305691e0-36c5-4b1a-ae4d-a2e43d4569cb%7D_Unity-Choosing_Netcode-Research_Report-v1_1.pdf

Unity documentation. 2022. Inverse Kinematics. Referenced 14.6.2022 Available at: https://docs.unity3d.com/Manual/InverseKinematics.html

Unity Multiplayer Networking. 2022a. Network Topologies. Referenced 6.2.2022 Available at : https://docs-multiplayer.unity3d.com/docs/reference/glossary/network-topologies

Unity Multiplayer Networking. 2022b. About Netcode for GameObjects. Referenced 20.5.2022 Available at : https://docs-multiplayer.unity3d.com/netcode/current/about

Valem. 2020. How to Make a VR Multiplayer Game – PART 2. Referenced 2.3.2022 Available at : https://www.youtube.com/watch?v=DB5bajOMdUQ

Wireshark. 2022a. Referenced 28.4.2022 Available at : https://www.wireshark.org/

Wireshark. 2022b. Start Wireshark from the command line. Referenced 28.4.2022 Available at : https://www.wireshark.org/docs/wsug_html_chunked/ChCustCommandLine.html