UNIVERSITY OF TARTU

Institute of Computer Science

Computer Science Curriculum

Sergii Shepelenko

Applying supernode architecture for scalable multiplayer computer game

Master's Thesis (30 ECTS)

Supervisor: Artjom Lind, MSc

Tartu 2017

Applying supernode architecture for scalable multiplayer computer game

Abstract:
Scalability, fast response time and low cost are of utmost importance in designing a successful massively multiplayer online game. The underlying architecture plays an important role in meeting these conditions. Peer-to-peer architectures, have low infrastructure costs and can achieve high scalability, due to their distributed and collaborative nature. They can also achieve fast response times by creating direct connections between players. However, these architectures face many challenges. Therefore, the paper investigates existing peer to peer architecture solutions for a massively multiplayer online games. The study examines two hybrid architectures. In the first one, a supernode approach is used with a central server. In the contrast in the second one, there is no central server and pure peer to peer architecture is deployed. Moreover, the thesis proposes a solution based on multicast peer discovery and supernodes for a massively multiplayer online game. Also, all system is covered with simulation, that provides results for future analysing.

Keywords: Games, Servers, Peer-to-peer, Game engine, Avatars, Hybrid architecture, Supernodes, Massively Multiplayer Online Game.
CERCS: P170

Supernode arhitektuuri rakendamine massiliselt online mitmikmängudele

Võtmesõnad:
Süsteemi skaleeritavus, kiire vastamise aeg ja madal hinnatase on tähtsad atribuudid, mida tuleb arvesse võtta suurte, multimängijatega online mitmikmängude loomisel. Sellistes süsteemides mängib suurt rolli arhitektuur. Partnervõrkude arhitektuuridel on madalad hinnad ning need suudavad saavutada järk-järgulise kasvu tänu nende hajususele ja koostööle. Peale selle suudavad nad kiirelt reageerida tänu otseühendustele mängijate vahel. Samas esineb selliste arhitektuuridega mitmeid probleeme. Selles lõputöös uuritakse olemasolevaid partnervõrkude lahendusi suurtele multimängijatega online olevatele mängudele. Veel uurib see lõputöö kahte hübriidarhitektuuri - esimeses on kasutatud supernode punkte koos keskse ühenduspunktiga ning teises on kasutatud keskset võrguharu ühenduspunkti ilma keskse ühenduspunktita. Lisaks sellele esitab see lõputöö lahenduse supernode multimängijatega online mängudele, mis põhinevad multiedastuse põhimõttel. Selleks, et tulevikus analüüse läbi viia, on kogu süsteem implementeeritud simulatsiooniga.

Võtmesõnad: Mängud, serverid, partnervõrk, mängumootor, hübriidarhitektuur, supernode'id, massiliselt online mitmikmängud.
CERCS: P170

Acknowledgements

I would like to express my gratitude to the following important people who have supported me during the whole Master's program. Firstly, I would like to thanks to my family, namely my mother Elena, my wife Olga and my sister Ann. You have all encouraged and believed in me.

I would also like to acknowledge and thanks to my supervisor Artjom Lind as well as to Amnir Hadachi. Thank you for valuable insights and recommendations that you gave not only throughout this thesis but during the last 2 years.

Finally, I would like to thanks to all my friends and especially to Vitalii Zakharov.

Abbreviations and Acronyms

MMOG - Massively Multiplayer Online Game
MMOFPS - Massively Multiplayer Online First Person Shooter
UDK - Unreal Development Kit
RPG - Role Playing game FPS - First Person Shooter
P2P - Peer-to-Peer
RMI - Remote Method Invocation
TCP - Transmission Control Protocol
AOI - Area of Interest
DHT - Distributed Hash Table
Rx - Received
Tx - Transmitted

Contents

1 Introduction

During the last two decades, the video game industry has grown from almost nothing, to a multi-billion dollar industry [16], where the best-selling titles generate more revenues than the biggest movie blockbusters [6].

Most modern games are online games, which allow players to challenge and/or cooperate with other players through the internet.The group of games known as massively multiplayer online games (MMOGs) is the most popular among gamers. MMOGs have taken advantage of the massive growth in the number of broadband users, in order to create digital online worlds, where thousands of players may participate.

Massively Multiplayer Online Games, like World of Warcraft or Lineage II have become increasingly popular in recent years. Today, millions of people meet and play in virtual worlds with friends and strangers.

Considering Massively Multiplayer Online games or First Person Shooters (MMOFPSs) games we can observe that the topology of the games in the most cases is based on Client Server architecture. The main problem in Client-Server network topology is that the performance of the game and session is mostly depend on the server itself. In case the server is overloaded, the latency is growing and the players have some problems with the game, which leads to players disconnection or even worst – losing the game items. However, when the number of players is increasing, the connection to the server will become a bottleneck, hence all players rely on one single point (Server side).

Jones et al. highlighted in [2] some business issues in the client-server architecture. This issues can be resumed to the large amount of hardware and employment of considerable staff to keep the servers running.

Various topologies have been suggested to eliminate the single failure point [5], or, leastwise decrease it. In this case, Peer to Peer networks is good choice to explore. Pure Peer to Peer type of network does not have a central server, and the bottleneck could be found only on the node itself, which is a part of the network.

The group of researchers from University of Malta have provided a survey [23] that was carried out around peer to peer networks, specifically, analysis of implementation of P2P for online gaming, primarily Massively Multiplayer Online

Role Playing Games (MMORPGs). Based on this, they were trying to get a system that is catered for online First Person Shooter (FPS) games. It will be shown below that such type of networks are not pure peer-to-peer, but such system combines Peer to Peer and Peer Supernodes use methods to bound the extensiveness of a server failure.

In this thesis, we wish to make a focus on applying Peer to Peer architecture for solving topology problems in first player style games. We will also investigate some existing solutions on how to handle network communication in games, some of which are used today and some of which are mere proposals. Lastly, we propose our own solution based on the peer-to-peer model with some extensions, that handle various current problems with the model.

Chapter 2 of the thesis starts with an overview of peer-to-peer architectures and the existing functionality as well as explains the problems each of them. The second chapter also introduces the existing solutions that try to solve similar problems. Moreover, we will get acquaintance few existing game engines.

Chapter 3 represents our proposed solution from the theoretical point of view as well as from implementation side. We will present a design for network architecture and will cover the most tricky parts of the implementation.

Chapter 4 depicts simulation results of proposed solution. Moreover, this chapter contains the test game application.

2 Background and Related Work

2.1 Overview of existing topologies

Before we proceed with applying supernode architecture and developing a multiplayer computer game, it is important to have an understanding of existing network topologies as well as game engines. Moreover, this chapter provides an overview of existing solutions that have been done in this area.

2.1.1 Centralized Architectures

Massively Multiplayer Online Games (MMOGs) usually apply Client/Server architecture (see Figure 1), where the server is used as a central point that hosts the virtual environment. The game is generated by a game host. This latter uses server for storing data related to the virtual world. When considering a client/server architecture it is assumed that there is the server that is responsible for processing tasks of the game node. From the other hand, the client is an application, that gives a player an opportunity to access a player instance and virtual world.

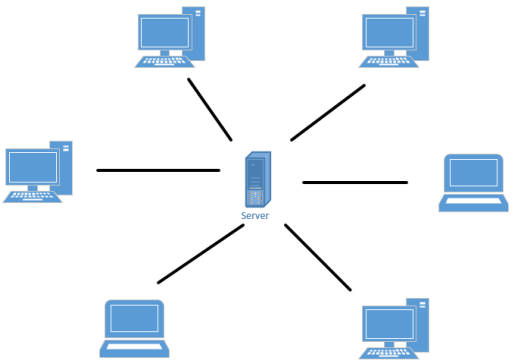

Figure 1: Client-Server architecture

Each client has its own account and should use its credentials for registering on the server in order to get an access to the game. In its turn, server grants an access to the virtual world, processes interactions between players, preserves game data, reports about periodic updates, controls the compliance of the game rules.

Usage of Client/Server architecture assumes that all player are connected to the server, hence server is responsible for processing all the game traffic.

It is possible to use several servers for the same game. However, in this case, it arises the problem of handling multiple players simultaneously. It is common to think that simultaneous interaction of all players is unreal when there is more than one server. Therefore as a solution, the virtual world of the particular game was divided in shards that are instances of that world and running at the same time. The purpose of each shard is to process part of the players. For example, let us consider World of Warcraft, the number of players constitutes around 6 million [1], and the players do not use one server. The game is played on more than 700 various shards spread across the world.

In scheme depicted above, the game traffic is divided between servers that are interconnected. It resolves the problem of a bottleneck in the network and allows to scale the game by adding new servers. In this case, hosting capacity is enhanced and there is no need to upgrade hardware.

Usage of the systems with Client/Server architecture, when a server is a central part of the system, gives privileges for a game host. First, it makes easier to do data back-ups because all data are kept on one server. Second, any changes in the game logic have to be done on the central unit that will affect the entire game. Moreover, such architecture enables to make the game more secure due to access management on the server, also player's actions can be checked by the server in order to find violations of the rules. Last but not least, in the case of growth of the players, the system can be expanded by upgrading the server hardware and Internet bandwidth.

Along with the advantages, there are drawbacks. The main problem is high expenses associated with hosting the server, maintaining and upgrading the hardware. Also, expenses related to bandwidth and usage of traffic. Furthermore, it is important to notice that apart from cost, there is also one huge problem that may happen in a centralized network topology, namely the game session heavily depends on the server itself. If the game traffic increases then the server may encounter overloading problem that leads to latency issues, which entails to session

[1] https://www.statista.com/statistics/276601/number-of-world-of-warcraft-subscribers-by-quarter/

disconnection or even losing the game data. The growth of players creates the bottleneck in the network. Hence, in such kind of architecture clients (players) are connected to one unit that is a central point of failure. In the case of failure, it may lead to data loss and as a result, it can cause the reduction of game players.

2.1.2 Purely Decentralized Architectures

Peer-to-Peer topology implies that every node in the network owns the same rights opposed to the Client-Server topology (see Figure 2). Applying the P2P architecture to the game will ensure that the game session is established even though the connection with one of the nodes is lost [1].

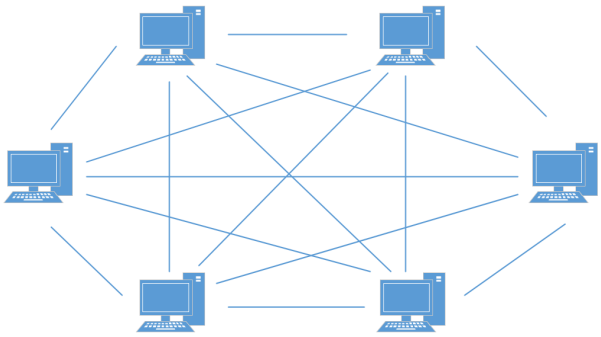

Figure 2: Peer to peer architecture

Usage of such architecture means that there is no central object which updates or keeps the global state of the game. Rendering the game and determining the game's global state are a responsibility of the player's machine. It is possible to consider the architecture as a huge computers grid, in which every component makes a contribution with some calculations.

Let's consider benefits of the Peer-to-Peer network. First is has a good scalability of the game due to the state of the game is computed on the client's machines, hence more game entities can join the game. Second, the usage of P2P topology eliminates the problem of having a singular point of failure (as was mentioned above, the game state is defined on player's side thus it is not preserved in the server). Thirdly, all nodes in the P2P network have an equal role. Hence, no server will face a bottleneck situation cause by overloading. Another advantage is that the clients can intercommunicate directly without increasing bandwidth

in contrast to the topology that has the server as a central point. Reducing the chance of bottleneck appearance in the network in its turn reduce latency in the network that leads to more stable connection.

Despite a lot of advantages, P2P architecture has some disadvantages that will be shown further. P2P is a fully connected network where nodes interchange messages for updating the state as well as for maintaining a consistent, shared sense of the game world. Such connectivity increases traffic growth, because of the rise in a number of clients and requires some optimization. For the player, it is necessary to be aware of the situation around its avatar. Resources on the player's side can be utilized for simulating a specific area called Area of Interest (AOI is a space that determines the surroundings near the player, which is visible to him) and for informing a player about the updates are occurred in AOI.

Also important to notice that P2P architecture may undergo cheating, because the global state of the game computed on the client's computer, hence the global state can be accessed and updated by the player.

When dealing with MMOG it is crucial to store the game state, because the connection of a player to the game is not necessary in order to update the virtual world. Moreover, the game should allow retaining player's data, for example, its possessions. The players assume that the game extracts the data between login sessions. In 2012 Gilmore and Engelbrecht published a research [11], authors have discovered that no current storage approach is well suited for a P2P MMOG.

The Figure 3 introduces the Peer-to-Peer topology. It is clear from the graph that all nodes are connected between each other to sustain consistency, also there is no central unit in the architecture that is responsible for global game state. Each peer stores and updates its part of global state.

2.1.3 Partially Centralized (Hybrid) Architectures

The hybrid architecture uses strong sides of the Client-Server and Peer-to-Peer topology in order to develop more efficient architecture. Hence, hybrid topology is used to make the game more scalable compared to centralized topology, at the same time keeping lower expenses to the game publisher and maintaining the required parameters. For example, control of the state of the game as well as applying

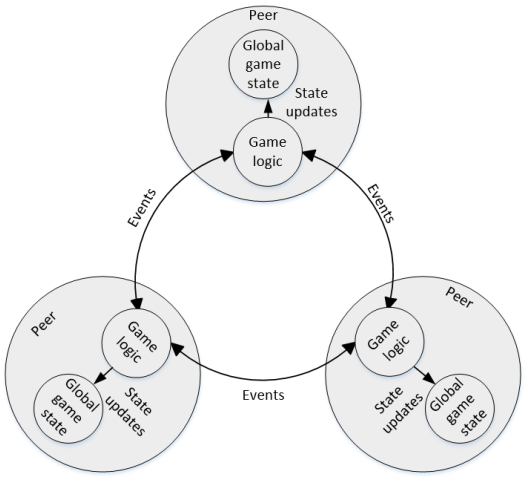

Figure 3: Peer to peer architecture in MMOG

simple methods for invoicing the players for the game items.

The structure of hybrid systems is decentralized as P2P, but on the other hand, it has features similar to the Client-Server architecture, such as nodes exist in the hybrid network with unequal rights. These nodes have more responsibilities than other nodes in the network and are called Supernodes.

In the hybrid architecture, central node is responsible for sustaining the consistent game state and preserving it, the server is also responsible for authentication, authorization, and content-distribution. Compared to the pure Client-Server topology such procedures need fewer resources, that leads to decrease in expenses for server hardware. Illustration of the Hybrid topology is depicted in Figure 4.

Usually, supernodes have more tasks to do (opposed to the other units), hence they require more computing resources. Supernodes behave similarly to servers in order to support standard nodes. However, one supernode is not responsible for all nodes, it is responsible for a part of them. Hence, all nodes in the network are united in the subsets and each supernode has its own subset of nodes.

As it was mentioned above supernodes have to be powerful, so central unit checks the computational resources of supernodes in order to find the most appropriate node to delegate rights, also server checks the throughput and latency. It is crucial to ensure the low latency because updates from central server go via the supernode to reach the client. Important to notice that supernodes will be chosen

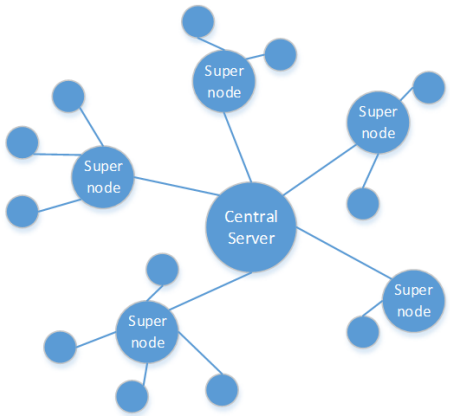

Figure 4: Hybrid peer to peer architecture

out of the nearest ones to the server. This is happened because of difficulty to measure the latency among clients and supernodes due to some players leave the game session and another join to it [10].

On the other hand, the hybrid topology has its own complexities. The network that uses such topology has to be able to add new nodes that are under control of a particular supernode. Also, the network load has to be split evenly between the supernodes. In case if one of the supernodes is unavailable, the system should redistribute the load to exclude the loss of information. Such systems have to be able to detect cheating notably if client plays the role of supernode. Moreover, systems that are using hybrid topology should grant motivation for the clients in order they make their resources accessible. This will ensure the low expenses [21].

2.2 Overview of existing game engines

As we are going to apply supernode architecture to MMOG, let's first of all review the state of the model game engine architectures. Based on the comparison, we will choose the most suitable for our purpose. In this chapter, we will observe widely used game engines like Unity, UDK, CryEngine, Cocos 2D-X as well as Monogame framework.

2.2.1 Unity

The Unity engine is a game engine with wide range of features, convenient and user-friendly interface. The main advantage of Unity is cross-platform integration, which makes it easily portable onto different platforms such as Android, iOS, Windows Phone 8, BlackBerry Windows, Linux, Mac. In addition, with the help of Unity, you can also develop games for PS 3, Xbox360, Wii U and web browsers[2].

The Unity engine supports sprites and 2D physics, letting developers create 2D games as well as 3D games.

The Unity engine does not support open source 3d models, neither it does not include a 3D model editor. Therefore, all the content must be created using a third-party 3D editor. However, the engine has an extensive library of 3D objects (models). Some of them are freely available, but most of they are have to be bought [12].

The Unity Personal Edition version is free, therefore is especially valuable for the low entry threshold for beginners. Moreover, a huge community around the engine has been organized. A low entry threshold is the result of a well-designed application: many things can be done with different editors, without writing a single line of code. Finally, Action Script, C# and JavaScript are supported by Unity [20].

The engine is written in C/C++ and has a closed code. However, it does not create any barriers, thanks to extended component structure and a wide range of supported languages.

2.2.2 UDK

Unreal Development Kit or UDK[3] is a free version of the Unreal Engine 3, which is used to develop many AAA games[4]. The bestsellers like Gears of War, Batman: Arkham Asylum, Mass Effect were developed using Unreal engine.

This engine has high graphics capabilities. Unlike Unity, the UDK has its own powerful tool for designing game levels. For the sake of simplicity the level editor

[2]https://unity3d.com/
[3]https://docs.unrealengine.com/udk/Three/WebHome.html
[4]https://en.wikipedia.org/wiki/AAA_(video_game_industry)

embedded to the engine itself [12].

The Unreal Engine was mainly designed to create FPS games but also is used to create games of a different genre, like RPG.

Similar to Unity, the UDK works with various platforms, including iOS, Android, Windows Phone 8, Xbox360, PS3, Playstation Vita and Wii U [20].

The Unreal has its own scripting object-oriented programming language, similar to Java or C++. Since many developers are using this engine, UDK probably has the best community among competitors. However, the engine has average entry threshold.

2.2.3 CryEngine

The CryEngine is an incredibly powerful engine developed by Crytek and first introduced in the first part of Far Cry[5]. With this engine, you can create games for PC and console, including Xbox 360, Xbox One, PlayStation 3 and 4, WiiU. As you can see, there is no support for mobile platforms, which is obvious considering the hight requirements for hardware.

The graphic features of CryEngine greatly exceed the capabilities of the Unity and UDK, because it includes state-of-the-art lighting, realistic physics, advanced animation system. The CryEngine has a stunning list of visualisation technologies, some of them: dynamic lighting and real-time shading, fogging, Terrain 2.5D, normal maps and parallax mapping, subsurface scattering, light rays and waves, control of the level of detail of the terrain, and much more other [20]. This is possible only on desktop platforms with a high-end graphical adapter, Therefore it obviously can't be supported by mobile platforms.

Similar to UDK, the GryEngine was a base for creating bestsellers like Ryse: Son of Rome. Compare to UDK and Unity the developers have to spend some time to understand it [20], thanks to its complicated design. Therefore we can mark this one with relatively high entry threshold. However, the engine has powerful capabilities for the level design.

In the absence of the source C++ code, you will steer the engine with the Lua scripting language, which is great for embedding into game engines, thanks to its

[5]http://www.crytek.com/cryengine

flexibility.

2.2.4 Cocos 2D-X

The original Cocos 2D[6] was developed in Python in 2008, next in the same year, it was ported to Objective-C under the iPhone (the version of Cocos 2D-iPhone). Two years later, a cross-platform version of Cocos 2D-X in C++ has been released.

Later, the versions with support for Android (Java), XNA (C#) and HTML 5 (JavaScript) appeared. Also, there is an extension for visualisation of three-dimensional graphics Cocos 3D, which allows making 3D models.

The following platform supports iOS, Android, Windows Phone 7 (XNA), BlackBerry, Tizen, Bada, Marmalade, Windows, Linux. The code can be written in C++, Lua and JavaScript.

The engine is used by monsters of the game industry like Zynga, Konami, Disney Mobile. Thanks to the frantic popularity of the engine (1.5 billion downloads of games based on it) the developers from Google, Microsoft, and Intel are participating in its development. On the other hand, the programmers-researchers are actively using this engine for quick development of visualization tools for scientific needs.

Accelerometer engine support allows you to create dynamic games like Hill Climb Racing [14]. The accelerometer is used to control the vehicle in the air.

There are several special individual editors (both paid and free) that aid in creating certain content: atlases, fonts, particles, sprite tables, etc. The set of GUI elements is rather small, but developers have left the possibilities can create missing components by themselves.

Despite the existence of strong community support, the engine is still a relatively complex development tool, therefore the engine has a relatively high entry threshold.

2.2.5 MonoGame

MonoGame introduces the cross-platform OpenSource implementation of the popular Microsoft XNA 4 framework, which is designed to work with graphics and

[6]http://www.cocos2d-x.org/

primarily for creating games.

In the past, Microsoft XNA 4 was allowing to use C# and VB.NET standard languages for the .NET platform to create complex visual scenes relatively quickly and easily. As a result development of three-dimensional games with rich graphics and sophisticated game scenarios. However, XNA 4.0 had one drawback - this platform, like all Microsoft products, was aimed exclusively at Windows.

Later, the project was renamed into MonoGame and moved to GitHub this made it open source. MonoGame became on platform natural, as a result, of rewriting it on OpenGL. This allowed developing games not only for Windows but also for Android, Mac and Linux platforms. Thus, the project began to move away from the orientation to one platform, which was inherent in XNA. As a result, today MonoGame supports the following platforms: Xbox 360, Windows (including the latest 10th version), Windows Phone, iOS, Android, Mac OS X, Linux[7].

The idea of the MonoGame platform is to utilize predefined game model. At regular intervals, certain methods are invoked, which updates the state of the game objects and redraw the objects in accordance with their new state. The user can also provide the input by clicking the mouse or the keyboard key to move the character. However, there is no click event. In the game cycle method, the state of each game object is polled. By expecting the state of the object we may discover if the movement was triggered. In this case, we update the state of the object and redraw the scene. Next, after the same interval, the game cycle method is invoked again.

The MonoGame game cycle can be represented in Figure 5:

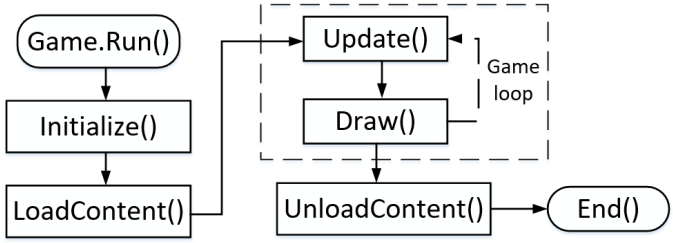

Figure 5: Monogame's game loop

[7]http://www.monogame.net/

1. In the Initialize() method, the game objects are initialized (the initial state of the game).

2. The LoadContent() method loads various resources into the application: pictures, audio files, models, avatars, etc.

3. Then the game cycle actually starts. With a frequency of 60 times per second (60 FPS), the Update() method is called and the game objects are polled. Here we find out whether the user pressed a certain key, whether two objects collided, etc. Also here we update the state of the game objects.

4. After the Update method, the Draw() method is also called 60 times per second (60 FPS) and redraws the scene. The control is then passed back to the Update method, and so on.

5. At some point, the user can press an escape key to exit the game. In this case, the exit the game cycle, and control is passed to the UnloadContent() method. In this method, the previously used resources are unloaded.

6. After that, the game ends and the application terminates.

This is the life cycle of any application on MonoGame, regardless of the type of project and operating system under which the game is written [17]. Further the Monogame framework will be used for rendering the game. This is simple framework that does not require a lot of time to get familiar with.

2.3 Introduction to ZephyrisNET framework

The group of researchers from Malta University have made an attempt to develop the framework [23], that allows deploying hybrid peer to peer architecture using supernode approach for multiplayer first player style online games.

The concept includes the main central server as an initial point of entrance to the game. The server is responsible for authenticating the players in the game. The idea was to give the opportunity to the companies, that aimed at creating a game, based on the central authentication point.

As a result, the players can be charged for some fee if he/she wants to buy some game items. This process is kept by the server control and authority. All other game computation the server delegates to the nodes, that will be introduced in the next paragraph. This decision allows saving expenses for server hardware.

On the other hand, the authors introduced the supernodes, that are responsible for all game states. Further, we will get acquainted with ZephyrisNET framework. We will find out what was the main issue, from the authors perspective and will familiarize with proposed solution.

2.3.1 Problem definition

The authors took as a basis Client/Server architecture and have tried to solve the common problems within this architecture. They understood when a game depends on a server it can happen that one failure is able to damage the entire gaming session, as a consequence it can cause the reduction in subscriptions. The authors came to a conclusion, that the most of the researches are focused on either peer to peer or hybrid peer to peer for MMORPGs [23]. This type of the game is more tolerant to higher degrees of latency in comparison with MMOFPS. In MMOFPS the extreme latency is able to cause the difference between a hit or a miss, so as a result, the reaction time in MMOFPS is crucial. Therefore, the authors set the main aim is to reduce latency in MMOFPS games.

2.3.2 Research

Before going deeper into developing the conceptual part of the framework, authors have got a visual representation of different types of games from the Claypool [15] and the latency effect on them.

Claypool [15] emphasizes that it is significantly important to understand latency effect, that can affect the game performance. In addition, the authors underline what type of games are more latency sensitive depending on player actions (quick or slow response). For instance, FPS games demand real time or almost real time response (if an action is slow it means that a player will be in the line of fire for a longer period). Claypool noticed that majority of RPG games is less sensitive to higher levels of latency. The Figure 6 depicts the dependency between

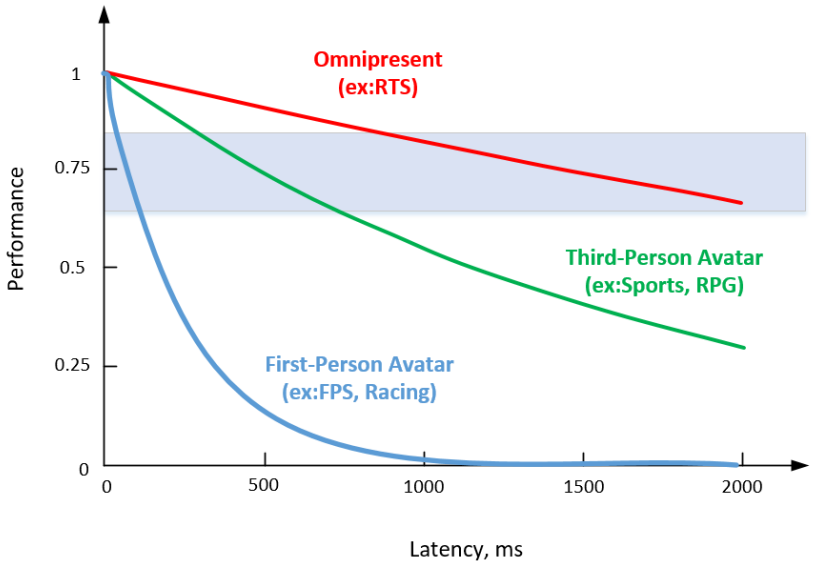

Figure 6: Player performance vs latency in different game types

latency in the game and performance depending on the game type.

Analyzing an architecture of P2P, hybrid models implemented in such MMORPG systems, authors are locking for a pattern based on "area of interest". Area of interest is located around avatar (a game entity introducing the player in the game world); it is a region that determines the surroundings near the player, which is visible to him.

Applying "area of interest" method, one is able to commence to "partition" corresponding game world depending on avatars, instead of server structure. For instance, in "Peer@Play" [4]. In "Peer@Play" the following paradigm was applied: all players are bound to each other in the P2P mesh. Communication among every "partition" is performed by choosing a peer (player) as a supernode [25]. Supernodes then communicate with each other thereby creating a large network of players.

Another technique was proposed by Pellegrino [8]. The idea was to use a central server in combination with P2P topology known as Peer to Peer with Central Arbiter (PP-CA). In this scheme, the players exchange updates in a P2P manner, but without carrying out verifications of consistency. The consistency of the game is verified by a CA, that obtains all updates, but gets in touch with

players in case if an inconsistency is discovered. However, the crucial problem with synchronization exists in this case. Therefore not only synchronization is needed to ensure consistency of the game state, the replication is able to assist with an incident of cheating P2P networks.

2.3.3 Methodology

Authors have suggested hybrid peer to peer system for a limited number of users, which relies on multicast for peer to peer communication. This system also allows the supernode to exchange information with a central server. However, a supernode is just a peer node that begins the session and creates the session ID that the rest peers will use to contact and therefore join the same session.

When an external server is obtainable, the supernode can set a connection. The person who is using the offered library can decide which information to transmit and which not.

Ultimately, authors have introduced a network library that is an open source, it is made with commonly exposed functions in mind, concealing the complication of P2P and Client-Server connectivity going on deep down.

2.4 Introduction to MMOG architecture based on a P2P overlay network

One more concept for Massive Multiplayer Online Games based on a Peer-to-Peer Architecture which is worth paying attention has been developed by the group of researchers from the University of Paderborn. The studies were focused on creating a peer to peer architecture for MMOG that allows supporting a large number of players distributed over the network [22].

To avoid a major drawback of peer to peer approach in the games, namely the lack of central server that regulates access to the game and prevents cheating, the authors inject the sets of controller peers that supervise each other [22].

2.4.1 Concept of MMOG architecture

To implement the system, that will meet all requirements of a distributed MMOG architecture, the authors proposed the system based on Pastry [18] and the extensions Past [19] as well as Scribe [9]. The overlay network consists of the next components:

1. DHT-based Overlay-Network – Pastry. The DHT-based overlay network should meet the requirements of self-organizing. The virtual infrastructure should allow providing the resistance to network failures. In addition, the infrastructure is highly scalable to maintain good performance with a large number of nodes. As MMOG requires some events for handling the game actions and players communication in comparison with traditional DHT-application for filesharing, the authors suggested Pastry [22] as a the most suitable for this.

2. For the object management the authors have used Past. To achieve the persistence of the objects and meeting high availability requirements with the object replication the authors have used Past system, which as well as a Pastry routes the messages over the overlay network [22].

3. For the event-based messaging, the authors have used Scribe. For spreading game action events to the other peers in the network the authors have chosen the Scribe system, which is multicast infrastructure for Pastry. It is obvious that event-based infrastructure should also satisfy the requirements of scalability.

The Figure 7 demonstrates how proposed solution works. On the first step, the object is found by Pastry itself. Next, it is necessary to synchronize all access. For this purpose, the Scribe is used. Finally, for storing the replicas in the current node leaf set of the logical network the Past system is used.

Authors emphasize, that all those technologies cover the requirements for distributed MMOG architecture.

The Figure 8 depicts the game engine that each peer has. The game engine is sitting on top of this network.

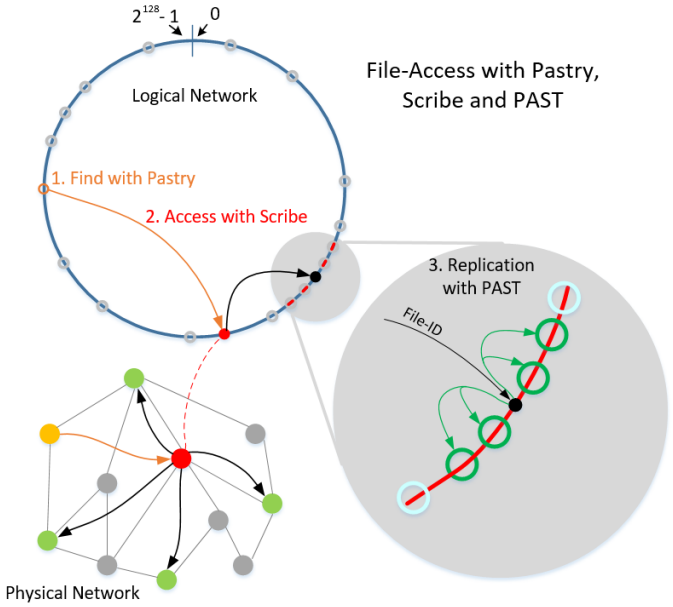

Figure 7: Synchronization and object access

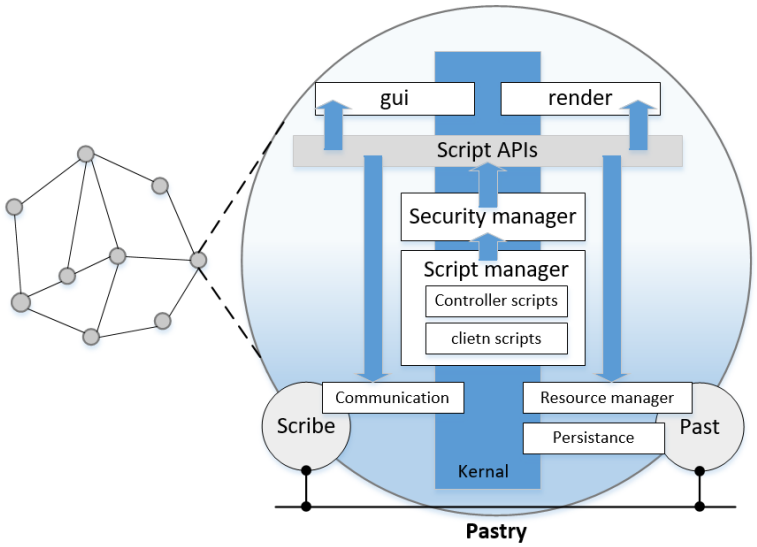

Figure 8: Single peer architecture

It is clearly observable, that a few components based on overlay network provide the functionality for a single peer node, namely:

1. The data manager stores game state as well as the data related to the client. It could be the login data, user profile data, the game avatars.

2. The resource manager stores the static data, like images, script source code, game objects.

3. The communication manager handles event spreading (through Scribe).

2.4.2 Network

The authors have segmented the virtual world into several parts. Each part is controlled by region controller (RC). The region controller is a regular peer that has been chosen from all peers, that are inside the current region. Moreover, the region controller is responsible for connection to other regions through portals or doorways.

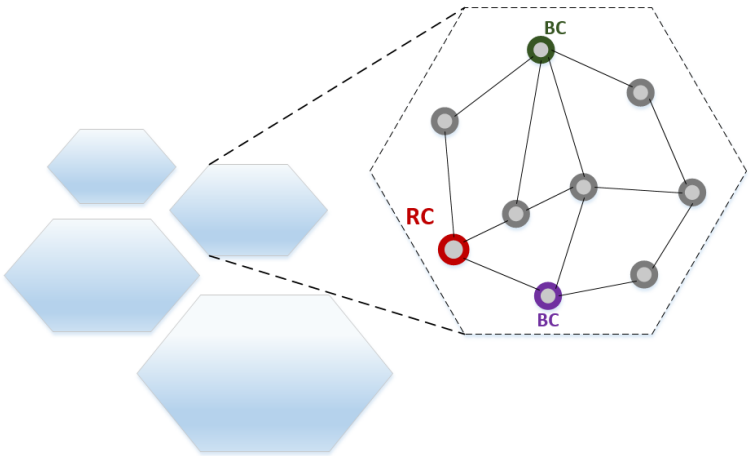

Figure 9: The region's concept in the virtual world

The Figure 9 shows the one part of the virtual world called region. Each region controlled by RC and running over a Pastry network.

As one of the main problems that can occur during developing of MMOG peer to peer game is to prevent cheating, the authors introduced the concept of backup controllers (BC). The main idea is to keep the copy of game state from region controller and in case if the last one will disconnect, the backup controller will immediately become a new RC. Moreover, the scheme with backup controllers allows recognizing when region controller has started cheating (see Figure 10).

Each peer in the region sends the event to region controller as well as to the backup controllers. Then all controllers calculate the game state based on events.

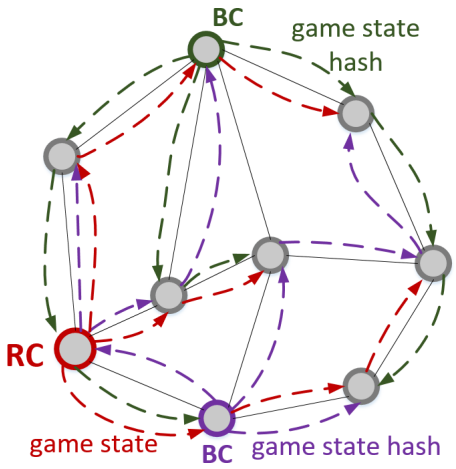

Figure 10: Cheating detection mechanism

Next, the region controller forwards new game state to all peers in the region. Simultaneously, the backup controllers send the hash value, that has been calculated from the new game state. The regular peer receives hash values from backup controllers and new game state from region controllers. Then the peer makes a comparison of those values. If the values match, then the new game state is received from region controller is considered as valid. Otherwise, the regular peer can determine which controller is sending wrong data. In this case, the controller may be hacked and based on the policy this controller should be expelled from the network [22].

3 Supernode architecture

3.1 Design and technology

In this section, I will introduce my solution. We will discuss my intermediate ideas and get a glance on how a supernode architecture can be applied to MMOG. Moreover, this chapter contains the description of the used technologies that we have used during implementation.

3.1.1 Network design

The main idea of this thesis is to create a hybrid peer to peer network that will use supernode approach. This means that some clients in the network will have more responsibilities than the others. Therefore, let's discuss this moment in more precise manner.

The most popular scheme for network management is zoning, where the virtual word is divided into smaller pieces called regions or zones (see Figure 11). Each region can differ in shape and level of difficulties. However, the player bounded within the region where this player is located. The player can simply interact with all players and game objects inside this region.

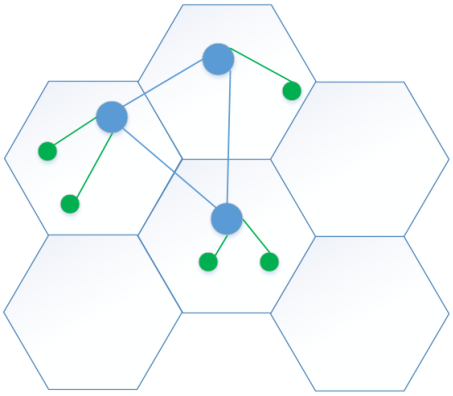

Figure 11: Area of interest

Moreover, to reduce the load on the server, the area of interest concept can be used. This allows sending only relevant stage changes in the game world (the closest territory) to the player. The Jean-Sebastien Boulanger [7] compares the

interest management algorithms that can analyze obstacles in the game and provide a minimum number of update messages between client and server (or client and client). However, in this thesis we will divide the game world into a number of game maps and for each player inside the current map, the area of interest will be entire map. It is performed for the purpose of demonstrating the supernode approach in a simple game.

From the network perspective the supernode is no more than a regular client that performs special computation allowing to process the input data from the other peers. However, the regular peers should know where to sent the data, that has to be processed. In this thesis, I will apply the supernode approach for MMOG. From this point of view, the supernode could be a client, that holds some piece of the game world (region) and takes all responsibilities for this area. Supernode approach supposes to organize some number of players in a separate network area where all outside communication from any player is accomplished via preselected peer (supernode). Moreover, this peer serves for all peers inside the region. All inside and outside communication are described in 3.3.2.

For outside region communication, in this thesis I will use the gate pattern depicted in the Figure 12.

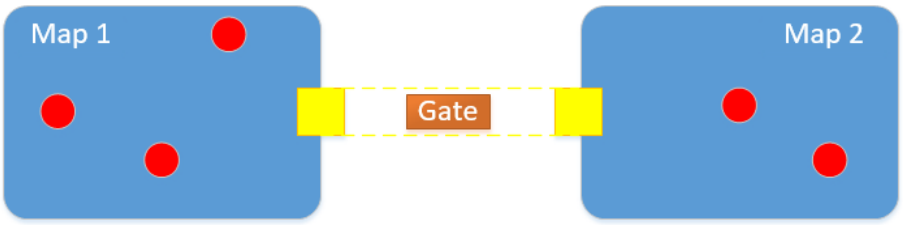

Figure 12: Gate concept

The idea is the following, in each map, there is a special game object that leads to the other map. When the player's avatar interacts with this object, the game starts the process of movement for this player from current map to another. This process is explained in details in 3.2.3.3.

Once we have decided with network topology we need to figure out how peers will discover each other. There were two network's designs that I used during implementation, namely hybrid peer to peer with a central server for authentication

and authorization as well as hybrid peer to peer without a central server.

The first topology includes the central server as an initial point for all clients (peers). The idea was the following, the client has to send a request to the server with some basic parameters. The server has to analyze the received request and based on the states of all maps and players provide the response to a newly connected client. During the analyzing, the server decides which role in the network the client will receive. It could be either the supernode or regular client. In case of the supernode, the server also assigns one map, that a newly client have to serve. Moreover, the server sends the list of available maps where the player can play, despite the client status (supernode or regular peer). The Figure 13 depicts the consistency of steps between client and server negotiations and subsequent attachment to existing supernode.

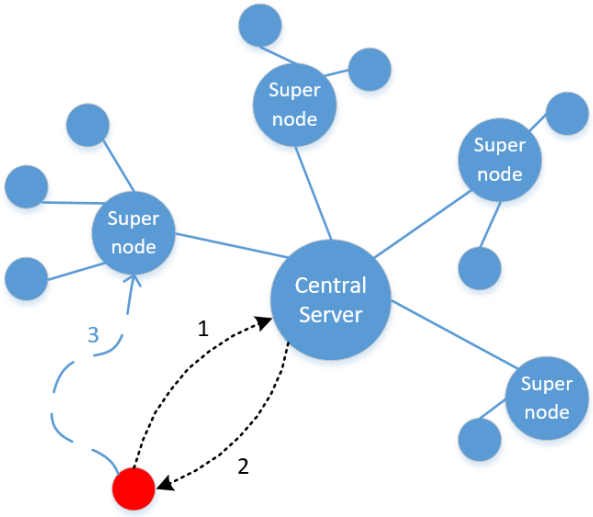

Figure 13: New player initialization in hybrid peer to peer architecture with central server

All other client's communication with all entities in the network goes through its supernode. More detailed client/server negotiations are described in the section 3.3.2.1.

From the other hand, when the central server is absent and there are no known peers for discovering the network, the newly connected client starts sending multicast packets to the predefined group (see Figure 14).

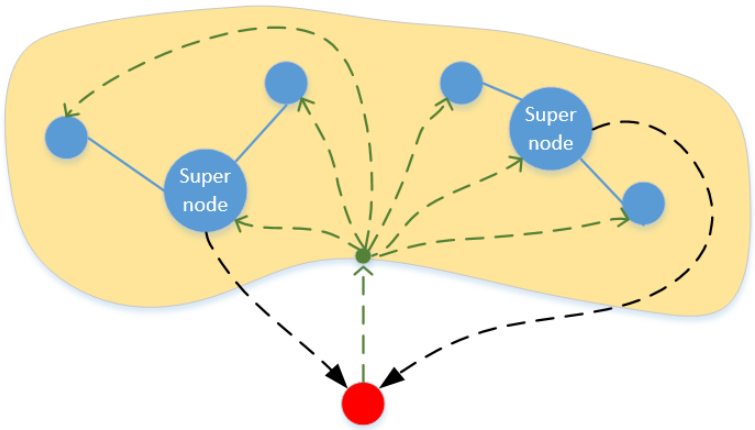

Figure 14: New player initialization in hybrid peer to peer architecture without central server

All nodes that are subscribed to this group listen to this channel and respond to received message based on message's type.

The idea is the following, the newly joined player sends a multicast message and received responses from all supernodes in the network. Then, based on embedded logic of supernode selection, the client analyzes the obtained results. Next, based on these results the new client can be either a supernode or a regular node. The supernode selection mechanism used in this thesis described in the section 3.2.2.

In conclusion, both topologies have its own advantages and disadvantage. However, in the case of a central server, all players initialization goes through this server. When the server is down, the new player cannot join the game, despite the fact, that all the players who joined earlier, before server failure are proceeding to play. Therefore, for our purpose, namely, to support scalability, we have chosen the hybrid peer to peer without a central server.

3.1.2 Used technologies

All code in this thesis has been written in C#. C# is an elegant, type-safe object-oriented language designed to develop a variety of secure and powerful applications running in the .NET Framework.

3.1.2.1 Monogame

In this thesis I am going to apply supernode approach for multiplayer game. For this purpose, I need to create a simple multiplayer game. From this point of view, Monogame framework is more suitable, from all previously described in section 2.2. Thanks to the Monogame framework it becomes easier to create a 2d tile-based game. To create and load tiles to the framework, we have used the Tiled editor that is described in next paragraph.

3.1.2.2 Tiled

Tiled[8] is a map editor that allows creating the content for your future game. Its main function is to edit tiles of different shapes as well as to support free image placement. A Tiled focuses on overall flexibility when trying to keep intuition (see Figure 15).

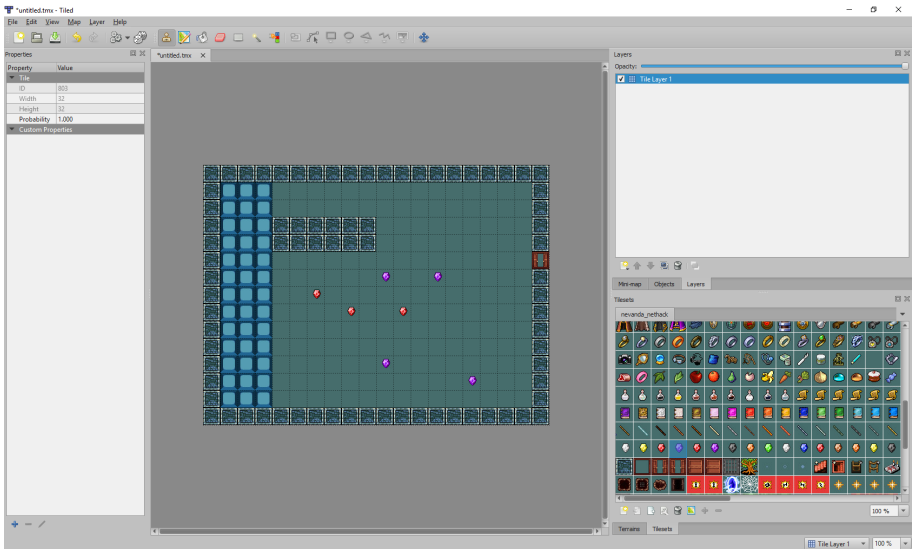

Figure 15: Tiled editor user interface

All you need is to create a tileset and load it to the editor. Then you can place any tile in different places creating the map with any level of difficulties.

[8]http://www.mapeditor.org/

3.1.2.3 .NET Remoting

.NET Remoting was developed to create distributed applications. With its help, you can access instances of .Net classes that are outside of your own domain (application domain). This can be another application within a single process, another process on the same machine, or a process on another machine (including the one connected via the Internet). .Net Remoting provides hidden possibilities like data marshaling, managing connections as well as reading and writing XML. .NET objects work with remote processes, which allows inter process communication.

The Figure 16 depicts how remoting mechanism works.

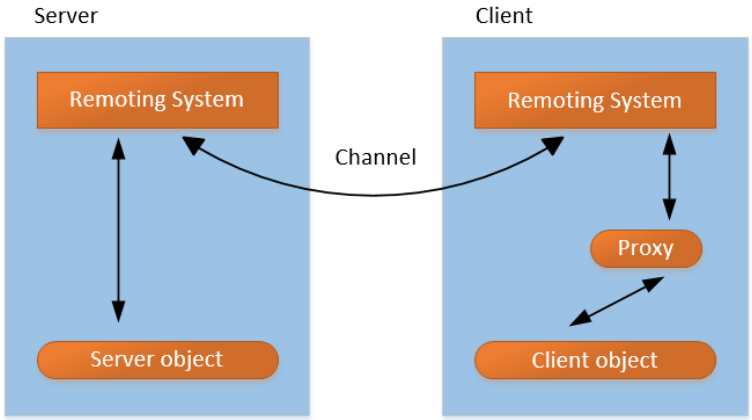

Figure 16: Remoting structure

In the beginning the server configures Remoting to use a specific protocol. This specifies the transport protocol and the access protocol. The system needs to create and register the channel on a specific port :

```
TcpServerChannel channel = new TcpServerChannel(port);
ChannelServices.RegisterChannel(channel);
```

The server then registers all the classes to which it provides access:

```
WellKnownServiceTypeEntry remoteObject = new
    WellKnownServiceTypeEntry(typeof(RemoteEngine), "remoteEngine",
    WellKnownObjectMode.Singleton);
RemotingConfiguration.RegisterWellKnownServiceType(remoteObject);
```

1. typeof(RemoteEngine) - registered class

2. remoteEngine - URI of the registered class. It is a parameter that the client uses to activate the object. Using an URI, the client tells the server that it needs an instance of the class RemoteEngine

3. WellKnownObjectMode.Singleton is an activation mode for each client call. Singleton objects serve multiple clients. Exchanging data is accomplished by preserving the information about status among client calls [13].

The client, in turn, must create a client channel and register the remote class in the local domain. The client, when accessing remote objects also specifies access protocols and sends a request to the server. The server in response to this request creates the requested object and passes its ID to the client. The client creates a special proxy class, which then uses it as if it were an object in its own domain.

```
var clientRemotingChannel = new TcpChannel();
ChannelServices.RegisterChannel(clientRemotingChannel);
RemotingConfiguration.RegisterWellKnownClientType(typeof(RemoteEngine
    ),"tcp://Ip:Port/RemoteEngine");
```

The last parameter in RegisterWellKnownClientType method specifies the location of the remote class. The protocol corresponds to the protocol of channels registered in application domains. Machine identifier is IP address that is set by the server that exports the RemoteEngine class and thus indicates the computer on which the object will be created. Moreover, the URI string indicates through a colon the port number on which the server expects calls.

At last, in order for the RemoteEngine class to support remote interaction, we have to use System.MarshalByRefObject as the base class:

```
public class RemoteEngine : MarshalByRefObject
{
...
}
```

3.2 Terminology

In this section, we will introduce the type of the nodes that are used in our network. We will familiarize how the system behaves based on the different game stages and

also we will figure out how supernode selection is implemented in this thesis.

3.2.1 Types of Nodes

During the research and implementation, I was dealing with 3 types of the nodes, each of them has its own responsibilities.

3.2.1.1 Central server

The central server manages entire network. All new players willing to join the game need to communicate with the server directly. As an entry point, the server provides information about the entities in the network, namely all maps where a new player can play.

Moreover, the central server controls all supernodes existence. Therefore all supernodes have to send keep alive messages to the server, thereby notifying the server about its presence in the network.

3.2.1.2 Supernode

The supernode acts as a "dedicated server" for the limited amount of regular peers. Moreover, it supports connections on demand with neighboring supernodes. Despite the fact, that a supernode has more responsibilities that regular peer, it still remains invisible for the player, meaning that all actions performed by a supernode are invisible for a customer.

3.2.1.3 Regular node

The regular node acts as a client in the client-server system. All regular nodes connected to the different supernodes and represent the player in the virtual world. However, in some cases, a regular node can become a supernode and take all corresponding obligations.

3.2.2 Supernode selection mechanism

Before we proceed, we need to figure out how regular peer becomes a supernode in other words we have to describe the supernode selection mechanism.

To select a supernode from a variety of peers it is a complex task that requires strong algorithms of analyzing the network and entities. The supernode should be well-dispersed throughout the peer-to-peer overlay network and satisfy such parameters as load balance, resource needs, fault tolerance, and heterogeneity [3]. Moreover, the supernode has to be stable in comparison with regular peers. A node that frequently joining and leaving the network, cannot fulfill supernode's requirements.

As the main goal of this thesis is to apply supernode architecture to multiplayer games I am not going to implement sophisticated supernode selection mechanism. It will be a simple mechanism, that cannot be used in production, but will be suitable for researching purposes.

The idea is straightforward, let's assume that our game consists of limited amount of game maps. Notice that we made an agreement earlier, that each map is served by one supernode. The mechanism supposes to create a new supernode from a newly connected client and assigns one of the available maps to it. For instance, a game has four different maps that have to be served by four supernodes. The first four connected peers will be assigned with those four maps. After that, all other players will appear as regular nodes in case of existence those four supernodes. The client which holds the supernode obligations is not limited in the choice where to play. The player can choose any maps on the game while proceeding to serve dedicated map.

3.2.3 Nodes Behavior

In this section, we will describe the nodes behavior in the network. To avoid the system failure it is important to define all nodes behavior in different situations. This will allow keeping the system consistent.

3.2.3.1 New node arrives

In order to start playing over network as a new player, the client application has to get all list of maps where the player can play. From the network perspective it could be either by sending a request message to the server or spreading the multicast message via the network. Once a client application has gotten the

response from the network, the player can choose a desirable map and invoke the start game process. From the the game side, the player appears at some location in the game world. It does not matter where. However, it should be predefined in advance. In this thesis, we have used one start position for a new player if this field is empty, otherwise, the position nearest to the start position. Of course, in this case, it might cause congestion of players, therefore for production purposes the other algorithm should be used.

Coming back to network's realization, on this stage of the game, the client application knows how it should act in the network either as a supernode or as a regular peer. Therefore, starting from here we can split our specification into two parts, based on node type. If the client acts as a regular peer, it begins communication with already known supernode. Otherwise, it acts as a supernode, which behavior described in next paragraph.

3.2.3.2 Supernode arrives

When newly connected client becomes a supernode, it means that this client has to notify all existing supernodes about its presence in the network. This is done through sending a multicast message to the group, that all supernodes are listening. All supernodes extract necessary information from the message, like which map is served by newly connected supernode or supernode's network location (IP address and port number). Moreover, each supernode activates a gate that leads to the new map. After that, the new supernode is ready to serve regular peers on this map.

3.2.3.3 Node's movements

All node's movements are served by a supernode. When a player moves inside the map, this may be considered as the inner movements. However, when player's avatar intersects with gate object, this is outer movements. Therefore, the current supernode starts direct negotiation with the supernode, where the gate leads. This process described in the section 3.3.2.2.

3.2.3.4 Node leaves

Handling leaving regular nodes is straightforward. The peer just needs to notify its own supernode about leaving. Hence, the supernode will delete this player from this map.

3.2.3.5 Supernode Leaves

Handling supernode leaving should be done by predefined procedure. This procedure assumes that the system has to find the substitution of this supernode from the players among those who are served by a supernode that is about to leave. For this purpose, the supernode has periodically sent the game stage to all the peer inside its map. Therefore, each peer can be potentially a new supernode when the real one will leave the game. This will allow to avoid interception in the game process. This process described in the section 3.3.2.3.

3.3 Implementation

3.3.1 Applications architecture

All the code in this thesis is divided into 3 applications:

1. WinForm application that serves as the client application. This application is responsible for the network and for all processes that are related to the network. All games actions or activities that are to be delivered over the network are handled by this application. Moreover, it provides a user interface for the game.

2. Monogame application which is responsible for rendering the game.

3. Shared library. Holds interfaces that can be implemented in the client's application allowing to embed the custom logic into game engine.

3.3.2 Communication Protocols

In this subsection, all communications protocols that have been developed in this thesis will be descried.

3.3.2.1 Client/Server communication protocol

This protocol has been developed for the case when there is the central server in the network. In order to join the game, the client first has to communicate with server to get all available maps. Therefore, the client sends his name on predefined server and receives a response packet from it (see Figure 17). When the

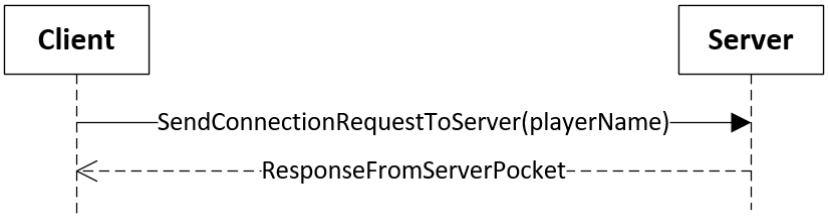

Figure 17: Client/Server communication sequence diagram

client receives a response from the server, it analyzes a content of the packet. The packet consists from the list of available maps and flag. The latter is responsible for a supernode mode. If this flag is activated, then the packet has one additional field that represents the map's name, for which this client will be responsible. After the client got a supernode's mode instructions, it starts running the engine in separate thread and all other clients will be served via this engine. For more information about engine see 3.3.3.

3.3.2.2 Supernodes communication protocol

One of the most important parts of the hybrid architecture is to define how and when supernodes interacts with each other. As were described in the section 3.2.3.3, the supernodes communicate with each other only in case when some of the players want to travel from one map to another, meaning that player's avatar already crossed the gate.

From the gamer point of view the procedure of changing supernodes is transparent. The only thing that player can observe is a message about changing map/level and "loading" label (see Figure 18). While the loading message is visible, all communications between supernodes are performed. The network entities do exchange some information and this activities are hidden from the client while he see the

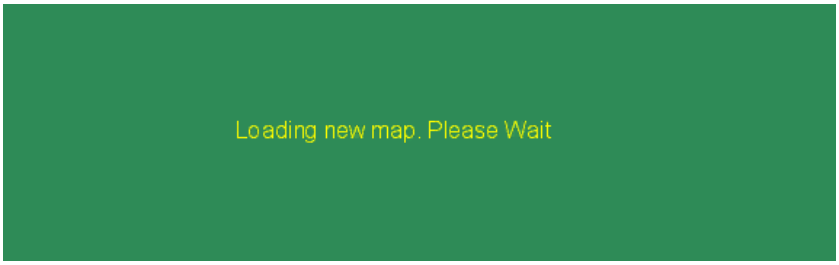

Figure 18: Loading map message

"loading" label.

After receiving the message, the current supernode preprocesses the data for sending to another supernode. The whole chain of player's transition from one map to another is depicted in Figure 19. Before current supernode starts communication with remote supernode, it puts the player object to its own Tx buffer and prepares to transmit the serialized player object over the network. Next, current supernode sends a request to a remote supernote. If a remote supernode has available room for this player, it sends back a positive confirmation and adds to the message player's ID. It allows the supernode to handle multiple negotiations with different supernodes.

When current supernode receives a response from the remote supernode, it sends a serialized player object to a remote supernode and waits for the confirmation of acceptance. Once the confirmation arrived, the current supernode deletes this player from its own list of players. On the other hand, the remote supernode puts the player to its own Rx buffer from where it can be added to the game.

After all negotiations steps are finished, the player's avatar appears on the new map, with preserving player's statuses (health, achievements, etc) and the loading message vanishes.

3.3.2.3 Supernode leaving protocol

In case that supernode is going to leave the game and at least one player is playing on this map, the supernode has to delegate its rights to handle this map to another player from this map. This is important in order to keep the game running for the other players. Therefore, the supernode has to find a substitution for itself

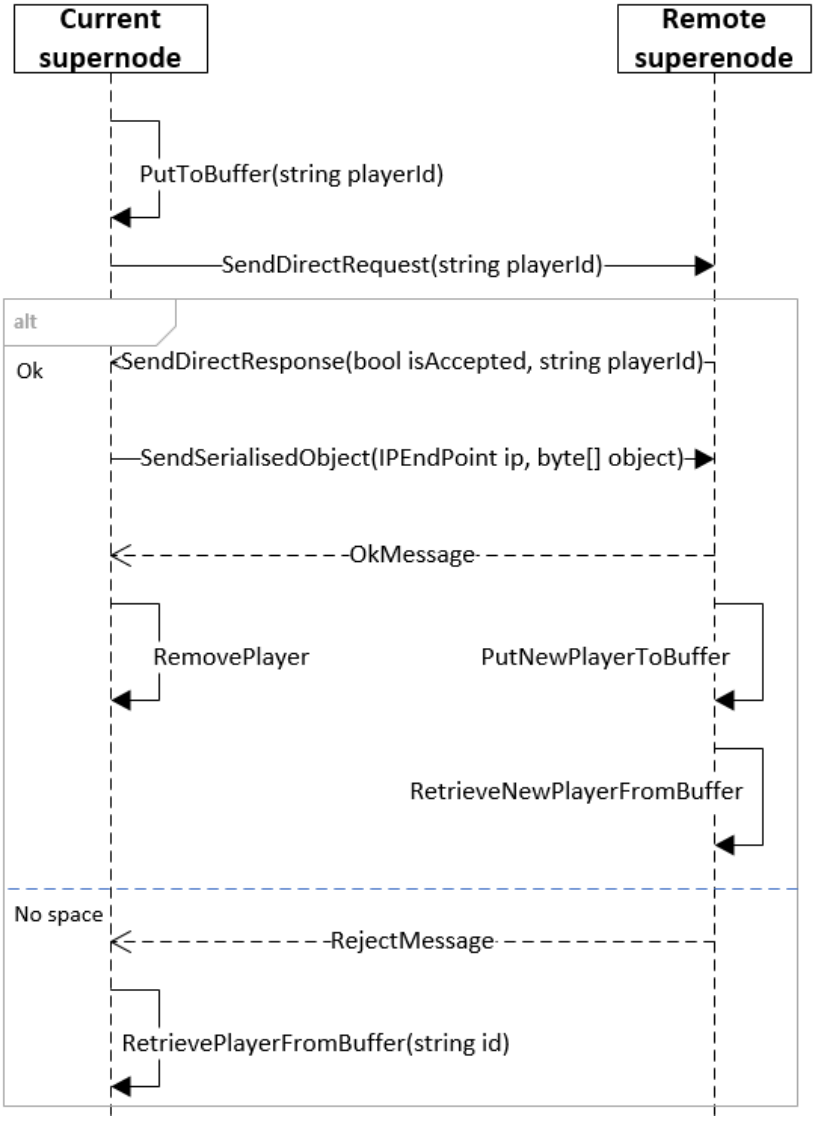

Figure 19: Supernodes communication sequence diagram

from its own list of players and notify all other supernodes about the substitution. The peers will experience small interruption with a message like "finding new server peer", however, in this case, all players will still have an opportunity to proceed a game on the same map.

3.3.2.4 Client/Supernodes communication protocol

In order to reconnect to supernode the client has to receive the proxy object of supernode's engine. Then, for each scenario of the game, the client has to invoke appropriate method for this action. If the player wants to move the avatar, the method like "set new player's position" has to be invoked. The engine will calculate and provide the results depending on the game logic.

3.3.3 Engine

Instead of keeping one engine for entire game on the server, supernode architecture allows to delegate game processing obligations to the supernodes. Each supernode responsible for one virtual world (map) no matter where this supernode is playing. It means that if supernode is playing out of its own map, it is still responsible for all gaming activities that are occurred within this map [24]. From programming perspective, the engine is an actor model, that has its own queue for handling requests (see Figure 20).

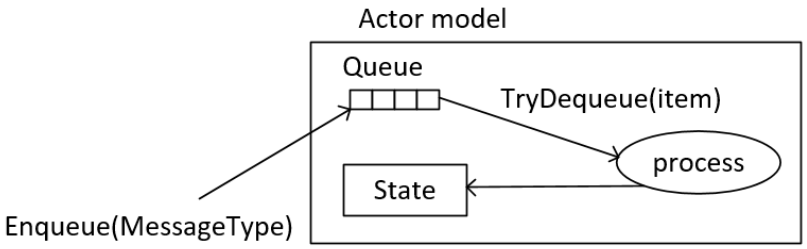

Figure 20: Game engine's queue

When the client needs to communicate with engine, it invokes remote engine's methods. Each of this methods creates its own message and puts it into the concurrent queue. Actor model processes the queue in separate thread and reacts on each message in accordance with certain rules [24]. To invoke remote methods, the remote method invocation technique has been used. The engine is responsible for all game process, therefore it can be customized depending on the purpose.

4 Validation and results

The main idea of this section is to describe how the system has been tested in this thesis. The section describes few test cases and provides the results. For the simulation purposes, an additional test application has been developed. It has been written in C# as well as all the code for this thesis. Additionally, this section provides an API, that can be used to deploy hybrid peer to peer system and apply it to the multiplayer game.

4.1 Simulations

The main goal of the testing is to measure the latency of the system. That means that we have to construct the system in the isolated environment as well as to reproduce the system's behavior. After that, we need to measure the time that system needs to process one request (hereinafter referred to as a ping). However, to get more accurate results the system should be under load. An isolated environment has been tested on a single machine. However, after that, the same tests over the real network were applied. All simulation parts allowed to get game's parameters under which the game keeps comfortable gameplay.

4.1.1 Isolated environment simulation

As mentioned above, those kinds of simulations have been accomplished on the single machine. The idea was the following: we have a supernode that handles one virtual world map as well as a number of regular nodes that connected to this supernode. Each node has to produce the same amount of load as a real system.

It may be assumed that in the real system each client sends a request to the supernode to get all necessary data N times per one second. The variable N can vary depending on game's FPS. Therefore, we tested the system with two different FPS, namely 30 and 60 frames per second. The design of test system is depicted in Figure 21.

For the isolated test system, we need to create a supernode and some number of regular peers (RP). All regular peers should produce the same load as in the real system, therefore each peer sends a few requests on each iteration. The number

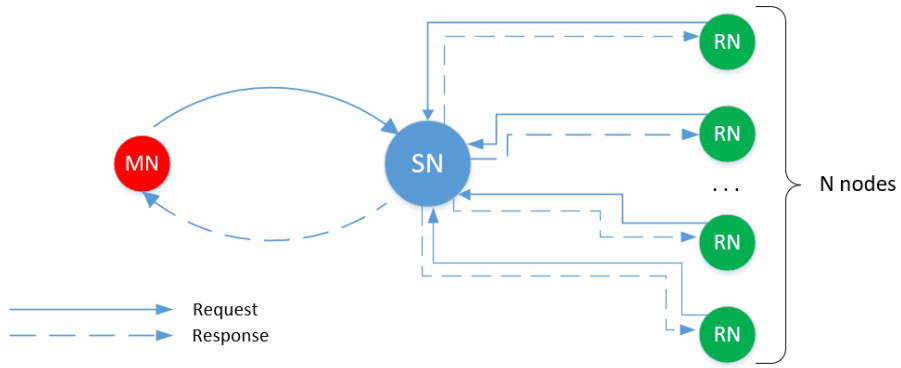

Figure 21: Test system

of iterations varies depending on FPS. The regular peer runs in its own separate thread. It allows testing system under the load.

When all required threads have started producing the load, we are introducing the measurer (MR). It acts as a regular peer, however, it sends only one request and measures the time of the response. The results of measurements are shown in Figure 22 and all measurements are presented in Table 1.

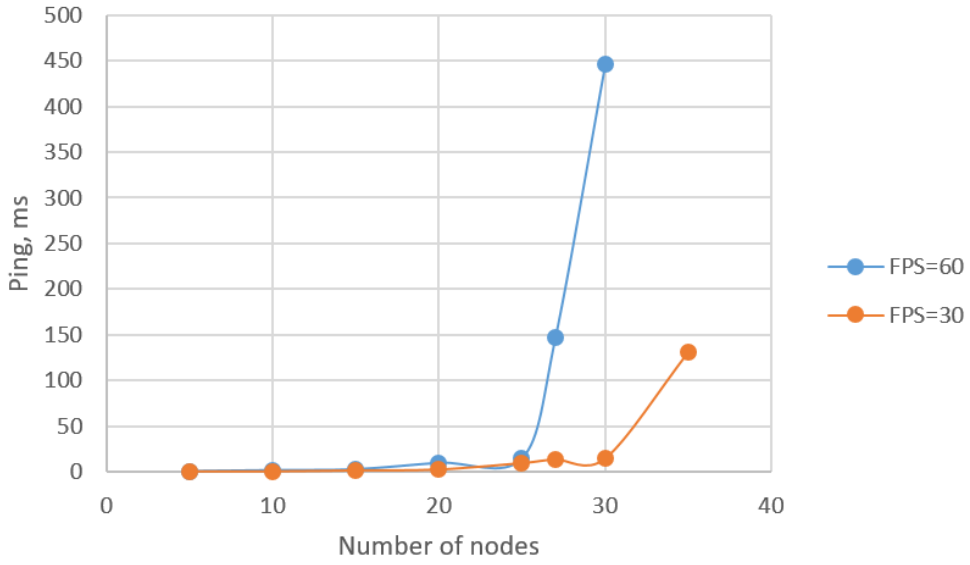

Figure 22: Dependency between number of regular peers and ping time in isolated system

From the graph, we can observe that the supernode serves for 30 and 25 regular

peers with FPS equal to 30 and 60 respectively. However, we have to take into account the characteristics of tested hardware, which had 8Gb installed RAM and Intel(R) Core(TM) i-5-5200U CPU processor.

Table 1: Dependency between the number of regular peers and the ping time in isolated system.

Number of nodes	Ping ms, FPS = 60	Ping ms, FPS = 30
5	1	1
10	2	1
15	3	2
20	10	3
25	15	10
27	147	14
30	446	15
35	1115	131
50	4163	1293
70	8994	4115

All the results above were obtained using an instance of the remote engine class directly. However, to get the tests closer to the real behavior, we need to repeat the simulation using RMI technique (.Net Remoting). The Table 2 presents the result of the simulation using RMI.

Table 2: Dependency between the number of regular peers and the ping time in isolated system using RMI.

Number of nodes	Ping ms, FPS = 60	Ping ms, FPS = 30
5	3	2
10	19	8
13	175	20
15	245	56
18	646	85
20	1441	140
22	-	340

From the Table 2 we can emphasize, that .Net Remoting framework adds la-

tency to our system. The total number of peers, that one supernode can serve is varied between 10 and 18 depending on FPS. Therefore we need to make additional tests and measure the latency when the load is split from one supernode into three supernodes. From the game point of view, this means that the players will cross from one map to the others. The results of the simulation are presented in Table 3.

Table 3: Dependency of the number of regular peers per supernode versus the ping time in isolated system using RMI.

	Peers per one supernode	Ping, ms	Total number of peers in the network
FPS = 60	4	29	12
FPS = 30	6	51	18

Reducing the load on one supernode in three times are not affecting on the same manner on the latency. However, the we can conclude, that the latency is reduced while the load is spread between supernodes.

In addition, we tested our system with players transition between maps. Massive multiplayer online game does not assume often transition between maps for players. Therefore we limited the transition in simulation to one minute. This means that each player can switch a map only once in a minute. The result for an isolated system with 2 supernodes was around 5 ms for transition of one player. The ping time is significantly less because the direct communication over TCP is used instead of RMI. In the next subsection, firstly, we will repeat our fist test over the network.

4.1.2 Network simulation

The network tests have been performed exactly on the same manner as the first isolated tests but only with one difference. The supernode and regular nodes were spread over the network. As a test environment, two same laptops and one router were used. It provides an opportunity to make a test over the network, that we could configure. However, during the network simulation, two networks were used,

namely, network with the Dlink router and the random local network (let's call it Network A). The results of all experiments are presented in Table 4.

Table 4: Dependency between the number of regular peers and the ping time over network.

Number of nodes	My network. Ping ms, FPS = 60	My network. Ping ms, FPS = 30	Network A. Ping ms, FPS = 60	Network A. Ping ms, FPS = 30
2	12	10	24	4
3	17	13	58	5
5	26	29	153	6
6	124	35	220	10
8	733	180	274	25
9	1540	407	1064	159
13	1624	1224	1289	874
10	1766	1853	2531	925
15	4233	4061	3133	1136
20	-	-	-	2464

Based on the results obtained over the network, we can conclude that the real network adds a latency to the system and response time is increasing. Moreover, we have tested our system on one network, where the nodes are located close to each other. Despite that, we can summarize that even with network latency, the supernode can process the requests from 8 regular peers simultaneously while preserving playability of the game. However, this only related to the games, which FPS is not greater than 30. If we want to expand the FPS to 60, then the supernode can serve only for 3 regular nodes.

Moreover, from the results, we emphasize that the response time also depends on the network configuration and productivity of the network equipment.

4.2 Results

This section introduces an API that can be used for deploying a massively multiplayer the game over our network architecture. Moreover, we will present an

overview of a simple game, that was used in this thesis.

4.2.1 Description of an API

The main idea of this thesis is to apply supernode architecture for a scalable multiplayer computer game. However, the system should allow to deploy the game using the method, which the engine exposes. Therefore, we are going to introduce the methods, that the developers can use in the custom client applications. As was mentioned in the section 3.3.1, the shared library application stores the interfaces that the client has to implemented in order to use the game engine. The list of engine's methods:

1. Name: CreatePlayer
   Signature: string CreatePlayer(int tileWidth, int tileHeight, PlayerType playerType)
   Return type: string value of player's ID
   Arguments: width and height of the player's tile and type of the player (it could be either the supernode or the regular peer)
   Description: This method creates a new player on the engine side, assignes a new ID to this player and returns a string value of the player's ID. This method is used when the new player connects to the game.

2. Name: AddPlayer
   Signature: void AddPlayer(IPlayer player)
   Return type: void
   Arguments: Takes an object of IPlayer interface
   Description: This method adds a newly joined player to current engine. This means that the player has been playing earlier, but only in the different map.

3. Name: SetPlayerPosition
   Signature: void SetPlayerPosition(string playerId, CustomVector2 newPosition)
   Return type: void
   Arguments: Takes an ID of the player and the desired new player's position

Description: This method allows to set a new player's possition in the virtual world.

4. Name: GetAllPlayersPosition
   Signature: void GetAllPlayersPosition (ICallbackAllPlayersPositions callback)
   Return type: void
   Arguments: ICallbackAllPlayersPositions callback
   Description: Provides the list of players' positions as a callback. This method takes as argument the object of ICallbackAllPlayersPositions interface. The client application has to implement the method GetAllPlayersPositions from this interface. The implementation of this method allows to save the data, that will be received through callback when the engine will process the request. After that, the client application can work with received data (for instance render all players).

5. Name: GetObjectStaticPositions
   Signature: void GetObjectStaticPositions (ICallbackStaticObjects callback)
   Return type: void
   Arguments: ICallbackStaticObjects callback
   Description: Provides the list of static objects positions as a callback. This method takes as argument the object of ICallbackStaticObjects interface. The client application has to implement the method GetStaticObjects from this interface. The implementation of this method allows to save the data, that will be received through callback when the engine will process the request.

6. Name: GetObjectsDynamicPositions
   Signature: void GetObjectsDynamicPositions (ICallbackDynamicObjects callback)
   Return type: void
   Arguments: ICallbackDynamicObjects callback
   Description: Provides the list of dynamic objects positions as a callback. This method takes as argument the object of ICallbackDynamicObjects.

The client application has to implement the method GetDynamicObjects from this interface in order to save the data, that will be received through callback when the engine will process the request.

7. Name: RemovePlayer
   Signature: void RemovePlayer (string playerId)
   Return type: void
   Arguments: string playerId
   Description: Removes player from engine's player list

8. Name: GetPlayerScore
   Signature: void GetPlayerScore (string playerId, ICallbackPlayerScore callback)
   Return type: void
   Arguments: string playerId, ICallbackPlayerScore callback
   Description: Provides the requested player's score as a callback. This method takes as argument the object of ICallbackPlayerScore and player ID. The client application has to implement the method GetPlayerScore from this interface in order to save the data, that will be received through callback when the engine will process the request.

Moreover, in this thesis, we used a gate concept. We embedded the gate entity into the engine, therefore the client application has to inherit the IGate interface from the shared library to allow the transition between maps. The Gate class that inherits IGate interface has to have 3 variables, namely string value of the map's name where this gate leads, boolean value that indicates either the gate is opened or closed and gate's position value that no more than coordinates of the gate.

Finally, we have to describe requirements of the map. As was mentioned in the section 3.1.2.2 in this thesis we used the Tiled editor for creating the maps. The map consists of an array of numbers, that refers to the field in the tile set. Here we have introduced additional character before each number in the array responsible for tile's type. The list of tile types are presented below:
1 - Floor object
2 - Wall object

3 - Player object

4 - Door/Gate object

5 - Dynamic game object (apple, diamond, etc.)

This guidance allows to use our hybrid peer to peer system and apply it to the multiplayer game.

4.2.2 Overview of proposed game

When you start running the application, the simple user interface appears (see Figure 23). From where we can enter player's name and initiate a process of

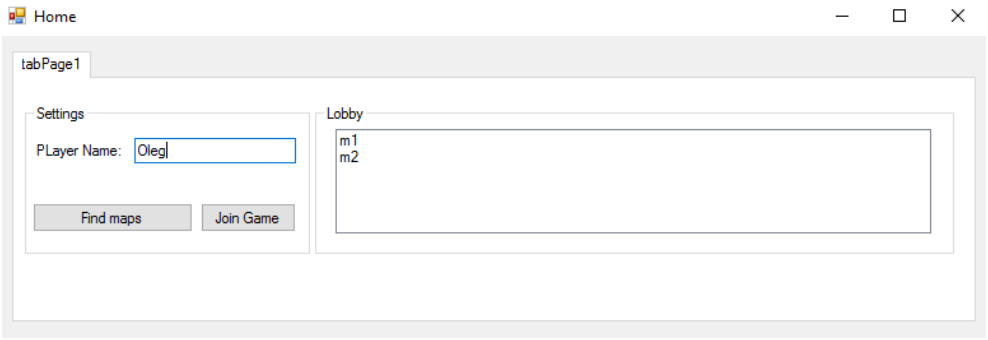

Figure 23: Game user interface

searching available maps. The lobby depicts all available maps from which the player can choose one map preferable for playing and press the join game button.

After the player pressed "Join Game" button, a new window with the gameappears (see Figure 24).

From the last figure, we can observe that this is one map with 2 players. One of this players is responsible for this map. The second player can be responsible for the second map or can be just a regular player. However, in this case, the second player holds the second map where the door leads. Moreover, there are the game objects like diamonds. When the player's avatar intersects it, the player score is increased.

To reach the second map, the player's avatar should intersect with door object. After that, the player will receive a loading map message which is shown in Figure 25.

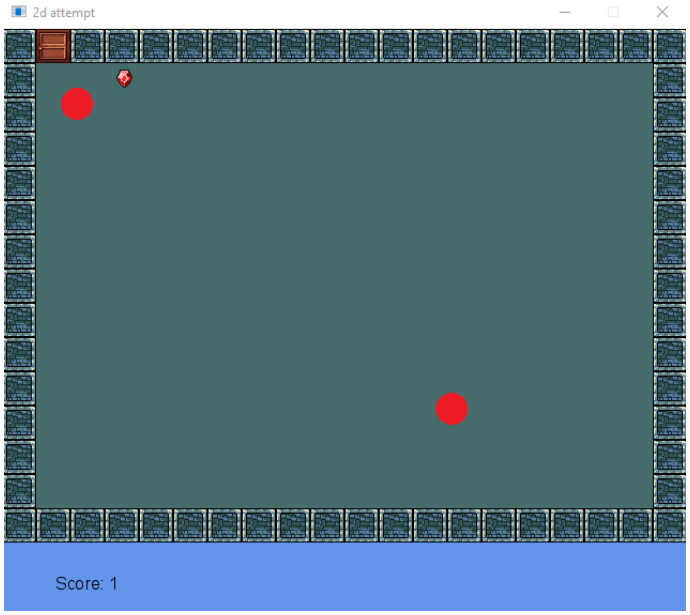

Figure 24: Game window

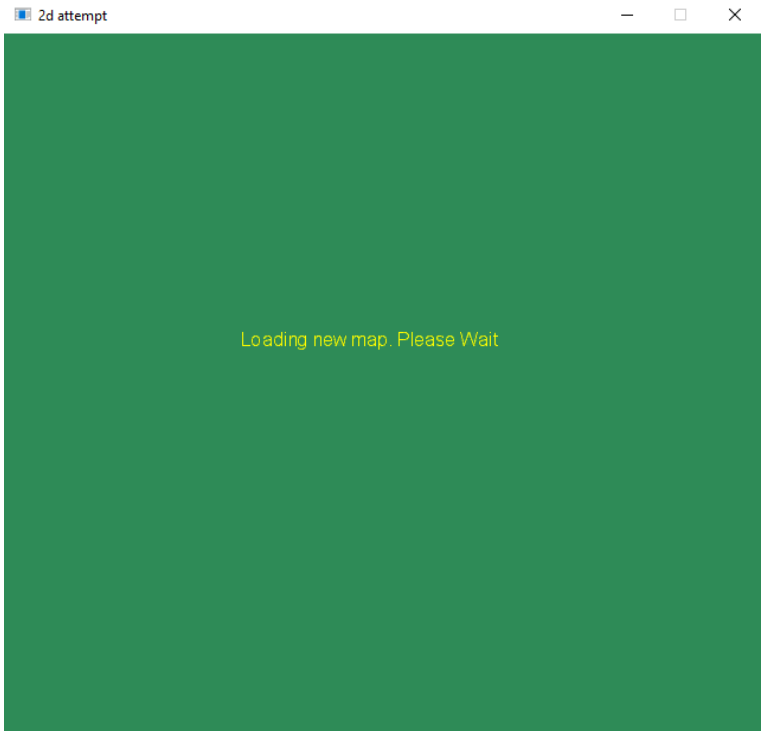

Figure 25: Loading map message

In brief, the whole game consists of multiple 2d maps that are interconnected through the gates. The gameplay is reduced collections diamonds by moving the player character through the map traversing the gates if nesesary.

5 Conclusion

This thesis investigates the ability to apply hybrid peer to peer architecture to a massively multiplayer online game. There are an impressive number of solutions how to deploy a massively multiplayer online game using peer-to-peer approach; however, mostly all of them remain in the early stage of development and do not go further. Moreover, it seems impossible to find an open source solution of a massively multiplayer online game using supernode approach that would be reasonable maturity level. Therefore one of the main goals of this thesis is to proof either this supernode concept can be applied to a massively multiplayer online game or not and to develop our own solution that will allow using our network layout for developing games.

As the players are sensitive to the latency in online games, we were looking for different ways of how to design our system. Moreover, we wanted to eliminate the central point of failure and bottleneck, therefore we used supernode approach without a central server.

Our proposed solution is based on the multicast peer discovery and handling the virtual world by a supernode. This allows to create a few number of virtual worlds independently while preserving a possibility to move between virtual worlds.

For an isolated system with direct access to the remote classes, the number of nodes varies between 25-30 nodes. However, with RMI technique the number of nodes is limited within 10-18. On the other hand, the simulations over the network with remote methods invocations showed that the network adds extra latency. Therefore, the number of peers have decreased to 5-8 nodes. Based on the results we can conclude that our system is not well developed for the real scenario and has to be improved in the future, hence, we can distinguish the few idea for the future work:

1. Increase the number of nodes that can be served by one supernode over the network.

2. Find the solution of how to reduce the latency using RMI.

3. Improve the algorithm of supernode selection.

4. Improve the algorithm of delegating supernode rights, when the latter leaving the game.

5. Make real tests over the network with at least few hops between nodes.

References

[1] Merabti M., El Rhalibi A. "Peer-to-peer architecture and protocol for a massively multiplayer online game". In: *Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE* (2004), pp. 519–528. DOI: http://dx.doi.org/10.1109/GLOCOMW.2004.1417631.

[2] Jones A. et al. "An analysis of peer to peer protocols for massively multiplayer online games". In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference* (2012), pp. 1994–1999. DOI: http://dx.doi.org/10.1109/TrustCom.2012.69.

[3] Lo V. et al. "Scalable supernode selection in peer-to-peer overlay networks". In: *Hot Topics in Peer-to-Peer Systems, 2005. HOT-P2P 2005. Second International Workshop on. – IEEE* (2005), pp. 18–25. DOI: http://dx.doi.org/10.1109/HOT-P2P.2005.17.

[4] Schiele G. et al. "Requirements of peer-to-peer-based massively multiplayer online gaming". In: *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium* (2007), pp. 773–782. DOI: http://dx.doi.org/10.1109/CCGRID.2007.97.

[5] Vitor de Albuquerque Torreao. "Comparison between client-server, peer-to-peer and hybrid architectures for MMOGs". In: (2013).

[6] Quinton Bronkhorst. *Games vs. movies: who wins?* URL: https://businesstech.co.za/news/general/19901/games-vs-movies-who-win/ (visited on 05/13/2017).

[7] Boulanger J. S., Kienzle J.P, Verbrugge C. "Comparing Interest Management Algorithms for Massively Multiplayer Games". In: *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games. ACM* (2006), p. 6. DOI: http://dx.doi.org/10.1145/1230040.1230069.

[8] Pellegrino J. D., Dovrolis C. "Bandwidth requirement and state consistency in three multiplayer game architectures". In: *Proceedings of the 2nd workshop on Network and system support for games* (2003), pp. 52–59. DOI: http://dx.doi.org/10.1145/963900.963905.

[9] et al. Castro Miguel. "SCRIBE: A large-scale and decentralized application-level multicast infrastructure". In: *IEEE Journal on Selected Areas in communications* (2002), pp. 1489–1499. DOI: http://dx.doi.org/10.1109/JSAC.2002.803069.

[10] Jardine J., Zappala D. "A hybrid architecture for massively multiplayer online games". In: *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games* (2008), pp. 60–65. DOI: http://dx.doi.org/10.1145/1517494.1517507.

[11] John S. Gilmore, Herman A. Engelbrecht. "A survey of state persistency in peer-to-peer massively multiplayer online games". In: *IEEE Transactions on Parallel and Distributed Systems* (2012), pp. 818–834. DOI: http://dx.doi.org/10.1109/TPDS.2011.210.

[12] Jamie O Flanagan. *Game Engine Analysis and Comparison*. URL: https://www.gamesparks.com/blog/game-engine-analysis-and-comparison/ (visited on 05/13/2017).

[13] Syed Nadeem ul Hassan. *.NET Remoting with an easy example*. URL: https://www.codeproject.com/Articles/14791/NET-Remoting-with-an-easy-example (visited on 05/13/2017).

[14] Sid James. *Top 10 Cocos2d-x Games Ever Made*. URL: http://blog.soom.la/2015/10/top-10-cocos2d-x-games-ever-made.html (visited on 05/13/2017).

[15] Claypool M., Claypool K. "Latency and player actions in online games". In: *Communications of the ACM* (2006), pp. 40–45. DOI: http://dx.doi.org/10.1145/1167838.1167860.

[16] Matt Matthews. *The Gamasutra Analysis*. URL: http://www.gamasutra.com/view/news/106211/NPD_For_July_2007_The_Gamasutra_Analysis.php (visited on 05/13/2017).

[17] *MonoGame Tutorials*. URL: http://rbwhitaker.wikidot.com/monogame-tutorials/ (visited on 05/13/2017).

[18] Rowstron A., Druschel P. “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems”. In: *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing* (2001), pp. 329–350. DOI: http://dx.doi.org/10.1007/3-540-45518-3_18.

[19] Rowstron A., Druschel P. “Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility”. In: *ACM SIGOPS Operating Systems Review* (2001), pp. 188–201. DOI: http://dx.doi.org/10.1145/502059.502053.

[20] Pluralsight. *Unity, Source 2, Unreal Engine 4, or CryENGINE - Which Game Engine Should I Choose?* URL: https://www.pluralsight.com/blog/film-games/unity-udk-cryengine-game-engine-choose (visited on 05/13/2017).

[21] Chen A., Muntz R. R. “Peer clustering: a hybrid approach to distributed virtual environments”. In: *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (2006), p. 11. DOI: http://dx.doi.org/10.1145/1230040.1230076.

[22] Hampel T., Bopp T., Hinn R. “A peer-to-peer architecture for massive multiplayer online games”. In: *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (2006), p. 48. DOI: http://dx.doi.org/10.1145/1230040.1230058.

[23] Cassar S., Montebello M., Zammit S. “Hybrid Peer to Peer and Server Client System for Limited Users Multiplayer First Person Style Games”. In: *Games and Virtual Worlds for Serious Applications (VS-GAMES), 2014 6th International Conference IEEE* (2014), pp. 1–8. DOI: http://dx.doi.org/10.1109/VS-Games.2014.7012161.

[24] Shepelenko S. *Design supernode architecture for computer games*. URL: http://ds.cs.ut.ee/courses/course-files/Sergii%20Shepelenko%20report.pdf (visited on 05/13/2017).

[25] Shepelenko S. *Hybrid Peer to Peer and Server Client System for Multiplayer First Person Style Games*. URL: http://ds.cs.ut.ee/courses/course-files/report-Sergii-Shepelenko.pdf (visited on 05/13/2017).

Appendix

A. Sergii Shepelenko. https://bitbucket.org/shepel07/seminargame Last Accessed: 18.05.2017

License

Non-exclusive licence to reproduce thesis and make thesis public

I, Sergii Shepelenko (date of birth: 5th of April 1988),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Applying supernode architecture for scalable multiplayer computer game

supervised by Artjom Lind

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 18.05.2017