# Design supernode architecture for computer games

Sergii Shepelenko
Computer Science
University of Tartu
Tartu, Estonia
sergii@ut.ee

***Abstract***
*Nowadays Multiplayer online games become very popular, mostly because the gamers have the opportunity to interact and compete between each other. Most of MMOGs is based on the Client Server architecture where players interchange periodic updates via Server. If the Server is overloaded, players start feeling the influence of network latency, which makes a game less playable. Therefore, the main idea of this article is to try to reduce the latency effects using Hybrid Peer-to-Peer architecture in Multiplayer online games* [1].

## 1. INTRODUCTION

In Client-Server architecture all players are connected to a central point of failure (server). It might arise a problem when the server becomes bottleneck in the network. When the server is overloaded, the latency is growing and the players have some problems with the game, which leads to player's disconnection or even worst – losing the game items.
When the number of players is increasing, the connection to the server will become bottleneck, hence all players rely on a one single point (Server side) [1].

## 2. RELATED WORK

This work is based on my previous research that I was working on during the Distributed Systems Seminar - Hybrid Peer to Peer and Server Client System for Multiplayer First Person Style Games. Hybrid Peer to Peer and Server Client System for Multiplayer First Person Style Games refers to the Malta's research group work where the main aim was to implement a simple library for peer to peer (and hybrid peer to peer/client server) game play system. The result of their work was release as ZephirisNET library written in .Net. The idea of this library was to deploy the hybrid peer-to-peer network structure and run the game over it using existing game engines like Unity, XNA, etc.
Also, they put forward idea, that the server conveys the management rights to supernode. Supernode is also a player (Client) which takes this responsibility from the Server and begins to manage sessions of other players. The network topology is depicted on Figure 1.

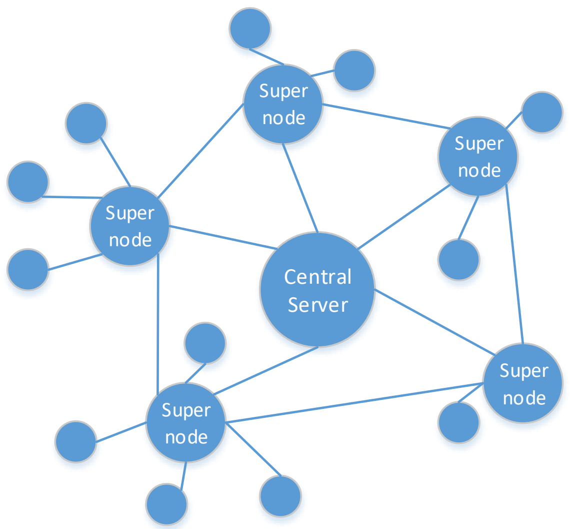

Fig.1 Supernode Architecture

Since the ZephyrisNET is an open source library, it was easy to combine what was announced and implemented. After some analysis it became clear

that ZephyrisNET library is a simple Client - Server architecture with some attempts to deploy hybrid peer-to-peer architecture.
ZephyrisNET showed that the system does not perform as expected, but it is still a good base for future work and expandability.
The results of the library show that it is possible to use it with the small amount of players, however there is no explicit distribution of responsibility between server and supernode.
The analyzed implementation of the ZephyrNET cannot compete with the description, which Shawn Cassar et al. have provided, therefore many questions remain open. Nevertheless, this work can be a good groundwork for future studies.
Based on previous research, we can specify the aims and goals for this work.

## 3. ACHIEVED WORK

Based on previous work I decided to implement my own version of supernode architecture and apply it on a simple multiplayer game. Considering these aims we can pick out two main parts like:

- Design and implement a simple 2D game;
- Implement a supernode architecture and deployment on it a 2D game.

### 3.1 Monogame Framework

For developing game, I have chosen Monogame Framework. It is an open source implementation of XNA that allows to create cross-platform games. If you are running the game that is written in Monogame in other platform, instead of running on Microsoft's .NET Framework, the games run on Mono.NET, which is an open-source port of the .NET Framework itself.
Now we can start designing the game and think about how it will behave in future. There should be some user interface where player can enter their data (name) and choose on which map he/she wants to play. Since I am going to implement all game and network architecture using C# therefore I can create a simple form for user interface. The approximate vision about it depicts on figure 2.

Client code also receives a notification if this client is a supernode or not. After that player connects to one of displayed maps and starts playing. For more detailed information about Client and Server communication see Client-Server Communication Protocol section.

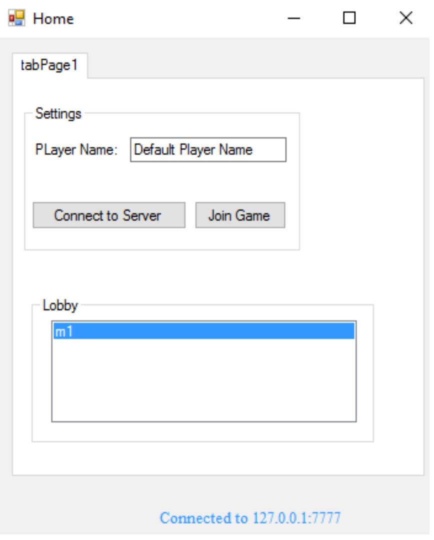
Home
tabPage1
Settings
PLayer Name: Default Player Name
Connect to Server
Join Game
Lobby
m1
Connected to 127.0.0.1:7777

Fig. 2 Game user interface

Whole game rendering process is managed by Monogame Framework. When the player presses “Join” button, user interface invokes game form (Monogame Framework) where game will be rendered (see figure 3).

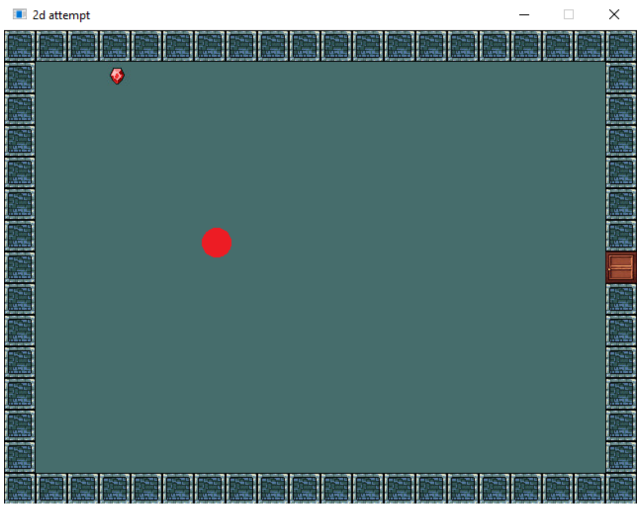

Fig.3 Game Form window in Monogame Framework

There are three main methods in Monogame Framework that allow to render the game:

- LoadContent – called once and loads all necessary (images, sprites, maps, fonts, entc);
- Update – updates all game objects
- Draw – draws all game objects

The Update and Draw methods run in game loop, so after Update is called, Draw is called, then Update, then Draw, and so on.

## 3.2 Game Logic

As we need to implement supernode architecture we assume that every map is served by one of the clients. It means that if a new client wants to play on this map, it has to establish connection with supernode. When client wants to change a map, its supernode has to request to the corresponding supernode (where client is going to move) and ask to connect a new player. Whole process you can observe in Supernodes Communication Protocol, which I am going to introduce in the next chapter. After all negotiations between supernodes are finished, the player can proceed his/her game in a new map with saving all the game states from the previous map.

## 3.3 Communication Protocols

### 3.3.1 Supernodes Communication Protocol

When the player wants to play in other virtual world, he needs to come to a specific game object known as gate. When one of the gates is triggered, the supernode understands that he needs to start negotiation with supernode where this gate leads. This process displayed on figure 4.
Before current supernode starts communication with remote supernode, he puts the player object to its own buffer and prepares this buffer to transmit the serialized player object over network after he recieves the confirmation from remote supernode. Then current supernode sends request to a remote and asks to take this player to his map. If a remote superenode has available room for this player, he sends back confirmation and new player Id. Also, remote supernode stores this player Id.
When current supernode receives response from the remote supernode, he assigns the player new Id and sends serialized player object to a remote supernode. Also, current supernode removes this player from its own players list.
On the remote side, supernode receives serialized object to its own buffer, than makes deserializing procedure. After that player proceeds his game in the other map.

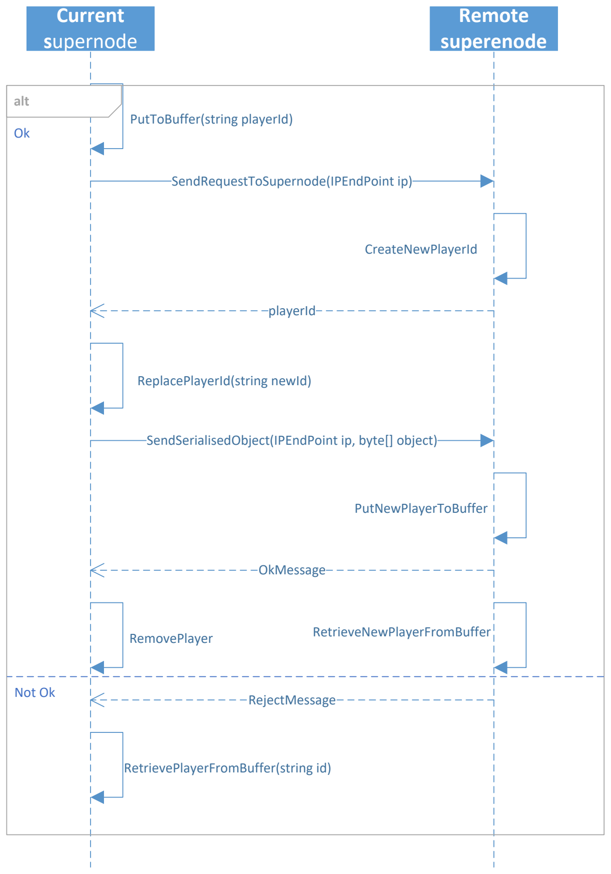
Current supernode
Remote superenode
alt
Ok
PutToBuffer(string playerId)
SendRequestToSupernode(IPEndPoint ip)
CreateNewPlayerId
playerId
ReplacePlayerId(string newId)
SendSerialisedObject(IPEndPoint ip, byte[] object)
PutNewPlayerToBuffer
OkMessage
RemovePlayer
RetrieveNewPlayerFromBuffer
Not Ok
RejectMessage
RetrievePlayerFromBuffer(string id)

Fig. 4 Supernodes communication sequence diagram

From the gamer point of view the procedure of changing supernodes is transparent. The only thing that player can observe is receiving a message about changing map/level and loading bar.

### 3.3.2 Client - Server Communication Protocol

In order to join the game client first has to communicate with server to get all available maps. Therefore, client sends his name on predefined server and receives a response packet from it (see figure 5).
When client receives a response from server, it analyses a content of the packet. The packet

consists from 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.

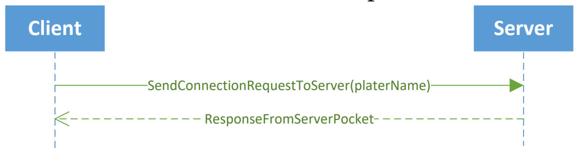

Fig. 5 Client - Server communication sequence diagram

Right after the client got a supernode mode instructions, it starts running engine in separate thread and all other clients will be served via this engine. For more information about engine you can find in Engine chapter.

### 3.4 Engine

Instead of keeping one engine for all game on server, supernode architecture allows to delegate game processing rights 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.
From programming perspective, the engine is an actor model, that has its own queue for handling requests (see figure 6).

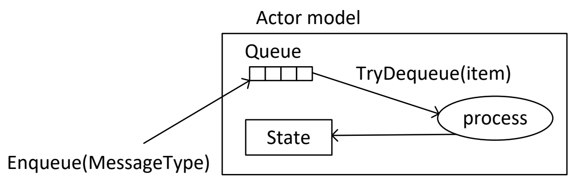

Fig. 6 Actor model

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. To invoke remote methods, I am using remote method invocation technique (.Net Remoting framework).

## 4. FUTURE WORK

The results of the library show that it is possible to add next features:

- Implement delegation of supernode rights. When the supernode wants to leave the game, it should delegate its rights to someone from his map. From the other hand, if there are no players, supernode can just leave and close the map.
- Implement some complex supernode selection mechanism.

## 5. CONCLUSION

In this research work I have developed supernode architecture. For the demonstration purposes I have implemented simple game. It is important to notice that system works as expected.
Supernode architecture reduces load balance from server, that makes game more reliable.

## 6. REFERENCES

1. Sergii Shepelenko "Hybrid Peer to Peer and Server Client System for Multiplayer First Person Style Games".
2. Shawn Cassar, Prof. Matthew Montebello, Dr. Ing. Saviour Zammit : "Hybrid Peer to Peer and Server Client System for Limited Users Multiplayer First Person Style Games"
3. R. Cecin, R. Real, R. D. O. Jannone, G. Martins, and J. Luis, "FreeMMG : A Scalable and Cheat-Resistant Distribution Model for Internet Games," *Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pp. 83–90, 2004.
4. G. Schiele, R. Suselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis, "Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming," *Seventh IEEE International Symposium on ClusterComputing and the Grid CCGrid 07*, pp. 773–782, 2007.
5. M. Claypool and K. Claypool, "LATENCY AND PLAYER," , p 40–45, 2006.